Trabajo Fin de Máster

NBA y sistemas de recomendación basados en *sequence modeling*

Álvaro Carro Viqueira

Máster en Técnicas Estadísticas

Curso 2025-2026

Propuesta de Trabajo Fin de Máster

Título en galego: NBA e sistemas de recomendación baseados en *sequence modeling*
Título en español: NBA y sistemas de recomendación basados en *sequence modeling*
English title: NBA and recommender systems based on sequence modeling
Modalidad: Modalidad B
Autor/a: Álvaro Carro Viqueira, Universidade da Coruña
Director/a: Jose Ameijeiras Alonso, Universidade de Santiago de Compostela
Tutor/a: Javier Soria Esponera, SDG Group; David Novoa Paradela, SDG Group
Breve resumen del trabajo: El presente TFM explora la intersección entre las metodologías de *next best action* (NBA) y *sequence modeling* aplicadas al ámbito de los sistemas de recomendación cinematográfica. El objetivo principal es evaluar el rendimiento de diversos modelos de aprendizaje profundo frente a enfoques tradicionales, desarrollando arquitecturas híbridas mediante la fusión de estos modelos.

Don Jose Ameijeiras Alonso, profesor titular de la Universidade de Santiago de Compostela, don Javier Soria Esponera, Specialist Lead de SDG Group, y don David Novoa Paradela, Senior Consultant de SDG Group, informan que el Trabajo Fin de Máster titulado

NBA y sistemas de recomendación basados en *sequence modeling*

fue realizado bajo su dirección por don Álvaro Carro Viqueira para el Máster en Técnicas Estadísticas. Estimando que el trabajo está terminado, dan su conformidad para su presentación y defensa ante un tribunal. Además, don Jose Ameijeiras Alonso y don Álvaro Carro Viqueira

√ sí ☐ no

autorizan a la publicación de la memoria en el repositorio de acceso público asociado al Máster en Técnicas Estadísticas.

En A Coruña, a 2 de junio de 2026.

El director:
Don Jose Ameijeiras Alonso

Firmado por AMEIJEIRAS ALONSO JOSE -

El tutor:
Don Javier Soria Esponera

El tutor:
Don David Novoa Paradela

David N.P.

El autor:
Don Álvaro Carro Viqueira

Declaración responsable. Para dar cumplimiento a la Ley 3/2022, de 24 de febrero, de convivencia universitaria, referente al plagio en el Trabajo Fin de Máster (Artículo 11, Disposición 2978 del BOE núm. 48 de 2022), **el autor declara** que el Trabajo Fin de Máster presentado es un documento original en el que se han tenido en cuenta las siguientes consideraciones relativas al uso de material de apoyo desarrollado por otros/as autores/as:

- Todas las fuentes usadas para la elaboración de este trabajo han sido citadas convenientemente (libros, artículos, apuntes de profesorado, páginas web, programas,...)
- Cualquier contenido copiado o traducido textualmente se ha puesto entre comillas, citando su procedencia.
- Se ha hecho constar explícitamente cuando un capítulo, sección, demostración,...sea una adaptación casi literal de alguna fuente existente.

Y, acepta que, si se demostrara lo contrario, se le apliquen las medidas disciplinarias que correspondan.

Agradecimientos

Quiero dar las gracias a mis tutores por hacer posible este trabajo. A Javi y a Jaime, por la propuesta de trabajo, así como por el apoyo y confianza depositados. A Jose, por sus aportaciones tanto a lo largo del desarrollo como en la redacción de la memoria. En especial, gracias a David, quien ha sido mi guía en todo este proceso. Gracias por enseñarme todo lo que me has enseñado, y por todo el empeño que le has puesto.

Por último, gracias a Irene y a mis padres por tanto.

Índice general

Resumen

Resumen en español

El presente Trabajo Fin de Máster explora la intersección entre las metodologías de *next best action* (NBA) y *sequence modeling* aplicadas al ámbito de los sistemas de recomendación cinematográfica. El objetivo principal es evaluar y comparar el rendimiento, la capacidad de personalización y la escalabilidad de diversos modelos de aprendizaje profundo frente a enfoques tradicionales. Asimismo, se desarrollan arquitecturas híbridas mediante la fusión de estos modelos avanzados con información proveniente de otros modelos de índole semántica o colaborativa.

A lo largo de la memoria se establecen las bases teóricas para cada arquitectura, junto con su respectiva formulación matemática. Al mismo tiempo, se muestra el proceso de implementación técnica: desde el previo ajuste de hiperparámetros de una red neuronal hasta la inferencia, pasando por la debida fase de entrenamiento. Las conclusiones se apoyan en métricas convenientemente establecidas para garantizar una comparativa lo más justa posible.

English abstract

This Master's Thesis explores the intersection between the methodologies of *Next Best Action* (NBA) and *Sequence Modeling* applied to the field of movie recommender systems. The main objective is to evaluate and compare the performance, personalization capacity, and scalability of various Deep Learning models against traditional approaches. Furthermore, hybrid architectures are developed by fusing these advanced models with information originating from other models of a semantic or collaborative nature.

Throughout the document, the theoretical foundations for each architecture are established, along with their respective mathematical formulation. At the same time, the technical implementation process is shown: from the prior hyperparameter tuning of a neural network to inference, including the corresponding training phase. The conclusions are supported by conveniently established metrics to guarantee the fairest possible comparison.

Prefacio

El presente documento repasa el Trabajo Fin de Máster realizado en modalidad de prácticas para el Máster en Técnicas Estadísticas con la empresa de consultoría SDG Group. Con el fin de asegurar la reproducibilidad y la escalabilidad del proyecto en el entorno empresarial, todo el marco metodológico y experimental ha sido desarrollado de forma nativa en Python.

Como firma especializada en la consultoría analítica y de datos, SDG Group se dedica a diseñar soluciones que optimicen la toma de decisiones y aporten valor a sus clientes en sectores de muy diversa índole. Por ello, los objetivos iniciales de este proyecto fueron combinar dos metodologías clásicas (*next best action* y *sequence modeling*) para desarrollar un caso de uso habitual y relevante para la compañía: la recomendación personalizada. La primera de las metodologías se centra en determinar cuál es la mejor acción que un sistema debe ejecutar a continuación, en un instante determinado. Por otro lado, la segunda es una rama del aprendizaje automático centrada en datos estructurados cronológicamente, donde la disposición temporal de los eventos es fundamental. El tema de la recomendación resulta de gran interés, pues permite investigar a fondo diversas arquitecturas de aprendizaje profundo. De esta manera, se ha logrado comparar modelos clásicos de recomendación como el filtrado colaborativo, contra modelos modernos como el Transformer, e incluso confrontar estos últimos contra arquitecturas completamente nuevas como xLSTM (2024) o ModernBERT (2025). Además, se han diseñado modelos híbridos mediante la combinación de distintas arquitecturas, con el fin de optimizar la capacidad de recomendación.

En el Capítulo 1 se justifica la necesidad de dar el salto desde los algoritmos clásicos de recomendación hacia el modelado secuencial. Se sientan las bases teóricas de las redes neuronales artificiales y el aprendizaje profundo, introduciendo formalmente el problema subycente de clasificación multiclase. A continuación, el Capítulo 2 se dedica al análisis y estructuración de los datos disponibles para entrenar los modelos. El núcleo del trabajo se halla en el Capítulo 3, donde se examinan en profundidad las diferentes familias de arquitecturas consideradas. Desde el algoritmo clásico Item-KNN, hasta los modelos de *sequence modeling*: Transformer, Mamba y xLSTM. Además, se detalla la metodología de integración para la construcción de los modelos híbridos, mostrando cómo enriquecer estas arquitecturas secuenciales con la semántica contextual de MiniBERT y ModernBERT, o con la topología estructural extraída mediante redes neuronales de grafos (GNN). Por su parte, el Capítulo 4 establece la metodología de entrenamiento y evaluación adoptada, definiendo las métricas empleadas. Finalmente, el Capítulo 5 recoge los resultados de la experimentación y expone las conclusiones generales del TFM. A través de una comparativa exhaustiva, se analizan los puntos fuertes y debilidades de cada modelo en términos de precisión, personalización y escalabilidad computacional. El documento concluye con una discusión sobre los desafíos y posibles líneas de investigación que abre este trabajo.

Capítulo 1

Motivación

El problema de la recomendación lleva décadas siendo relevante. Hoy en día nos rodea por completo: cada vez que entramos en una plataforma de *streaming*, navegamos por una tienda online o consultamos nuestras redes sociales, estamos interactuando con sistemas previamente diseñados para captar nuestros gustos y/o necesidades. Estos sistemas responden a una pregunta fundamental: ¿qué contenido es más probable que sea de interés para cada usuario?

Un *sistema de recomendación* utiliza, en general, tres tipos de datos (véase Ricci et al., 2015): ítems, usuarios y transacciones. Los ítems son los objetos que pueden ser recomendados. Dentro de un mismo sistema, estos deben compartir un "tipo" específico: libros, películas, noticias, productos, trabajos, etc. Por otro lado, los usuarios pueden ser muy distintos entre ellos. Para captar sus gustos personales, es posible considerar información complementaria como datos demográficos. Finalmente, se denomina transacción a una interacción registrada entre usuario e ítem. Esta podría expresarse con variables binarias (que constatan si un usuario ha interactuado o no con un cierto ítem) o cuantitativas, como la calificación asignada a un artículo dentro de una escala numérica.

Desde una perspectiva algo más técnica, es posible definir un sistema de recomendación mediante el objetivo de estimar la siguiente función de utilidad,

$$f : U \times I \longrightarrow R,$$

donde U e I son los conjuntos de usuarios e ítems respectivamente, y R representa un conjunto (en general) totalmente ordenado. La función estimada $\hat{f}$ permitiría obtener una puntuación para cada par (u, i). De esta manera, la recomendación para un usuario u podría consistir en ofrecer aquellos ítems i_k con la puntuación $\hat{f}(u, i_k)$ más elevada.

1.1. Enfoque

Tradicionalmente, los sistemas de recomendación han sido clasificados en dos categorías principales:

- **Filtrado colaborativo:** Se trata de ofrecer recomendaciones a un usuario basándose en ítems que han sido consumidos por otros usuarios con gustos similares. Por ejemplo, si dos usuarios A y B muestran un interés elevado por los libros de terror, a cada uno se le recomendarán aquellos títulos que aún no ha visto, pero que sí han sido consumidos por el otro. Es la técnica que más ha evolucionado en los últimos años, debido al auge del aprendizaje profundo.
- **Filtrado por contenido:** El sistema trata de recomendar ítems similares a otros que haya disfrutado ya. Por ejemplo, en una plataforma de compras online, si un usuario ha comprado o

valorado positivamente zapatillas deportivas de una determinada marca o estilo, el sistema le recomendará otros productos con características similares, como zapatillas de la misma marca, diseño o rango de precio.

Sin embargo, estos enfoques han sido siempre estáticos: las recomendaciones se generan a partir de historiales fijos, sin tener en cuenta la evolución de los gustos del usuario en el tiempo ni el contexto en el que se realizan las interacciones. Para superar estas limitaciones, han surgido enfoques más avanzados. Uno de ellos es el ***sequence modeling*** (o *sequence-aware recommendation*, véase Quadrana et al., 2018), que considera el comportamiento del usuario como un historial ordenado, captando la naturaleza cambiante de sus intereses. Las nuevas arquitecturas surgidas a partir de modelar el orden cronológico inherente a los datos han sobrepasado a los modelos estáticos en la mayoría de los ámbitos.

Por otro lado, el concepto de ***next best action*** (***NBA***, véase Cao y Zhu, 2022) se centra en determinar cuál es la siguiente acción que un sistema debe tomar con el objetivo de optimizar la experiencia de usuario. En el ámbito de la recomendación, esto podría traducirse a la predicción del próximo ítem que el usuario desearía consumir: *next best item* (*NBI*). Combinar este concepto con el *sequence modeling* permitirá crear modelos más inteligentes y contextuales, capaces de adaptarse en tiempo real al comportamiento del usuario (véase la Figura 1.1).

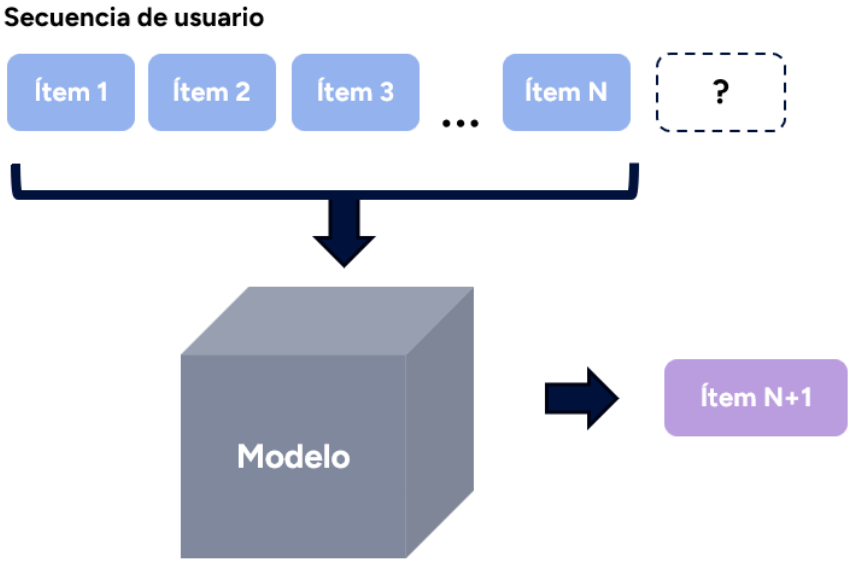

Figura 1.1: Ejemplo del funcionamiento (en inferencia) de un modelo de recomendación secuencial, aplicando el concepto de NBI. El modelo recoge un historial de usuario ordenado, lo procesa e intenta estimar el próximo ítem a consumir.

Como se tratará en la sección a continuación, este enfoque moderno se basa en utilizar modelos de aprendizaje profundo (redes neuronales con gran número de capas), subrama del aprendizaje automático. En particular, se establecerá un enfoque de *clasificación multiclase*: cada ítem será tratado como una clase independiente.

El primer objetivo de este trabajo (véase la Sección 1.3) es construir un sistema de recomendación cinematográfico. De esta manera, se necesitará un conjunto de datos que contenga historiales ordenados de consumo de estos ítems (películas) por parte de cada usuario. En este caso, las transacciones serán las valoraciones que cada usuario haya dado a las películas vistas, y el trabajo será intentar predecir la próxima película que verá el usuario (véase la Figura 1.2).

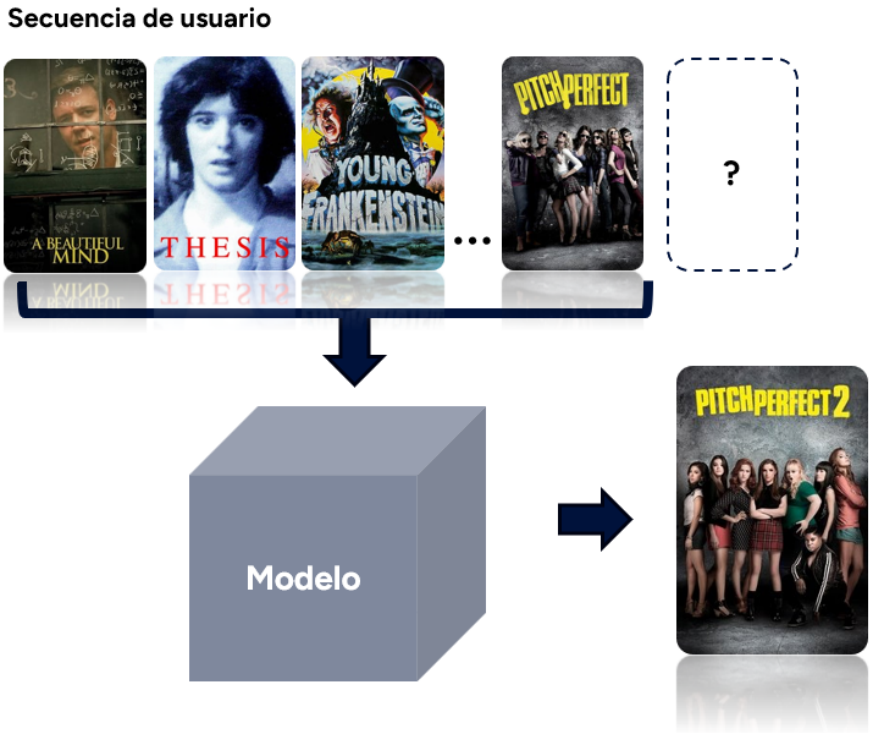

Figura 1.2: Ejemplo del funcionamiento de un modelo de recomendación secuencial de cine. Cada película ha sido valorada por el usuario siguiendo un orden concreto (historial de transacciones). El objetivo del modelo es recomendar una nueva película, maximizando la probabilidad de que el usuario la disfrute. Las carátulas de las películas se incluyen con fines puramente ilustrativos y académicos, siendo propiedad de sus respectivos titulares de derechos de autor. Imágenes obtenidas a través de la API de The Movie Database (2026).

1.2. Introducción a las redes neuronales artificiales

Las redes neuronales artificiales son modelos de aprendizaje automático inspirados en el funcionamiento del cerebro humano. Están formadas por un conjunto de nodos simples y conectados llamados *neuronas*. Cada neurona recibe una señal proveniente de todas sus conexiones previas, procesándola y enviándola a las próximas neuronas (véase la Figura 1.3). Veamos cómo se desarrolla esta "sinapsis":

1. Dado un vector de entrada $\boldsymbol{x} = (x_1, \ldots, x_d)$, la neurona calcula

$$z = \boldsymbol{w}^\top \boldsymbol{x} + b,$$

donde $\boldsymbol{w} \in \mathbb{R}^d$ es un vector de pesos y $b \in \mathbb{R}$ es un sesgo. Estos parámetros son inicializados de manera aleatoria y se aprenden en una fase de *entrenamiento*.

2. Posteriormente, se aplica sobre z una función de activación σ:

$$y = \sigma(z),$$

obteniendo la señal final. Algunas funciones de activación habituales son la sigmoide,

$$\begin{aligned} \text{sigmoide} : \mathbb{R} &\to (0,1) \\ z &\mapsto y = \frac{1}{1+e^{-z}}, \end{aligned} \tag{1.1}$$

la ReLU

$$\begin{aligned} \text{ReLU} : \mathbb{R} &\to [0, \infty) \\ z &\mapsto y = \text{máx}(0, z), \end{aligned} \tag{1.2}$$

o incluso la SiLU

$$\begin{aligned} \text{SiLU} : \mathbb{R} &\to \mathbb{R} \\ z &\mapsto y = z \cdot \text{sigmoide}(z) = \frac{z}{1 + e^{-z}}. \end{aligned} \tag{1.3}$$

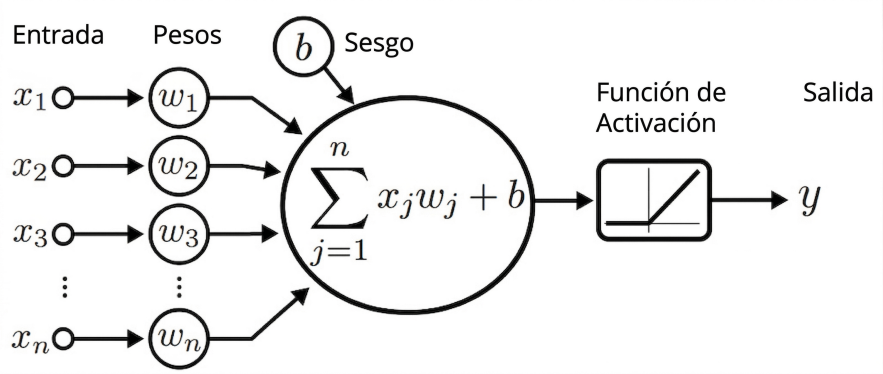

Figura 1.3: Esquema del funcionamiento de una neurona.

El orden en el que se conectan las neuronas sigue una estructura de capas formadas por varias neuronas con entradas $\boldsymbol{x} = (x_1, \ldots, x_{d_x})$ y salidas $\boldsymbol{y} = (y_1, \ldots, y_{d_y})$. En una red neuronal completamente conectada con L capas, cada una de ellas transforma la salida de la capa anterior. Si denotamos por $\boldsymbol{y}^{(l)}$ a las señales de la capa l (también llamadas *activaciones*), entonces

$$\boldsymbol{y}^{(l+1)} = \sigma\left(\mathbf{W}^{(l)}\boldsymbol{y}^{(l)} + \boldsymbol{b}^{(l)}\right) = \sigma\left(\boldsymbol{z}^{(l)}\right),$$

donde $\mathbf{W}^{(l)} \in \mathbb{R}^{d_{l+1} \times d_l}$ es la matriz de pesos y $\boldsymbol{b}^{(l)} \in \mathbb{R}^{d_{l+1}}$ el vector de sesgos, con d_l, d_{l+1} las dimensiones de las capas correspondientes (es decir, el número de neuronas, que no tiene por qué coincidir). Nótese que, con el objetivo de compactar la notación, los pesos para cada "sinapsis" están ahora recogidos en una matriz donde cada fila se corresponde con un vector de pesos $\boldsymbol{w}^{(l)}$ para la entrada $\boldsymbol{y}^{(l)}$ (y de igual forma los sesgos $b^{(l)}$). La función de activación σ se aplicará sobre cada componente de $\boldsymbol{z}^{(l)}$. Por otro lado, la dimensión de estos vectores dependerá de la naturaleza del problema.

En este caso, la información de los usuarios (es decir, los vectores de entrada $\boldsymbol{x}$ y salida $\boldsymbol{y}$) se podría codificar mediante vectores de dimensión $|I|$ (siendo I el conjunto de ítems), donde cada posición representa una película del catálogo. Estos vectores tomarían el valor 1 en los índices correspondientes a los ítems visualizados, y 0 en el resto de los casos. De esta manera, el vector $\boldsymbol{y}$ será un vector de ceros salvo en la posición que ocupa el último ítem visualizado en el vocabulario de películas. Se verá en la Sección 3.2 que existen maneras más sofisticadas de codificar esta información, a través de los llamados *embeddings*.

Como se observa en la Figura 1.4, la primera capa de neuronas $\boldsymbol{x} = \boldsymbol{y}^{(0)}$ se denomina *capa de entrada* (para la introducción de los datos), las intermedias $\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(L-2)}$ son las *capas ocultas* y la capa final $\boldsymbol{y}^{(L-1)}$ se llama *capa de salida*. Es importante escoger bien la función de activación para esta última capa. Como estamos ante un problema de clasificación multiclase, los elementos de la salida $y_i^{(L-1)}$ deben ser asociados a una probabilidad. Es decir, ser positivos con suma 1. De esta manera, se

define la función de activación SoftMax como sigue:

$$\begin{aligned} &\text{SoftMax} : \mathbb{R}^{|I|} \to [0,1]^{|I|} \\ &\qquad \boldsymbol{z} \mapsto \boldsymbol{y}, \ \text{con } y_i = \frac{e^{z_i}}{\sum_{j=1}^{|I|} e^{z_j}} \quad \forall i = 1, \dots, |I|. \end{aligned} \tag{1.4}$$

De esta manera, la salida y_i se corresponde a la probabilidad de que el próximo elemento a consumir por el usuario sea el ítem i, condicionada al resto del historial.

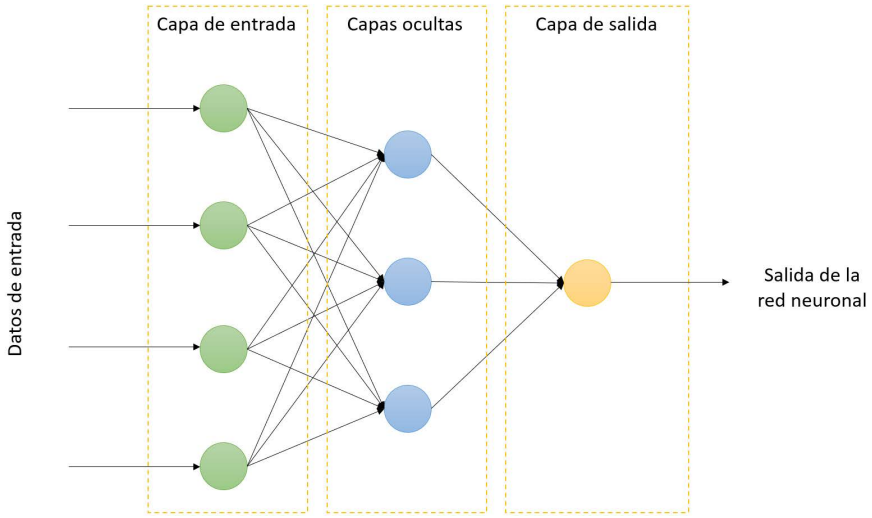

Figura 1.4: Esquema de una red neuronal simple con una única neurona en la capa de salida.

Entrenamiento e inferencia

Una red neuronal precisa una fase de *entrenamiento* con datos seleccionados para ello. Además de los datos de entrada $\boldsymbol{x}$, se deben proporcionar unos datos de salida $\boldsymbol{y}$, con los que poder comparar la estimación final de la red, $\hat{\boldsymbol{y}} = \boldsymbol{y}^{(L-1)}$. El objetivo al entrenar una red neuronal es encontrar los parámetros $\mathbf{W}^{(l)}$ y $\boldsymbol{b}^{(l)}$ de cada capa, que minimicen una función de pérdida determinada $\mathcal{L}$, adaptada para cada problema y arquitectura. Debido a la naturaleza del problema de clasificación, la función a minimizar será (salvo en un caso que se verá en la Sección 3.6) la entropía cruzada (*cross-entropy loss*),

$$\begin{aligned} &\mathcal{L}_{\text{CE}} : \{0,1\}^{|I|} \times (0,1)^{|I|} \to [0,\infty) \\ &\qquad (\boldsymbol{y}, \hat{\boldsymbol{y}}) \mapsto \mathcal{L}_{\text{CE}}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\sum_{i=1}^{|I|} y_i \log(\hat{y}_i) = -\log(\hat{y}_{\boldsymbol{y}}), \end{aligned} \tag{1.5}$$

donde $\boldsymbol{y}$ son los datos originales e $\hat{\boldsymbol{y}} = \boldsymbol{y}^{(L-2)}$ su estimación. El vector $\boldsymbol{y}$ valdrá 1 en la posición de la película correcta que vio el usuario, y 0 en el resto. Así, la función de pérdida es simplemente el logaritmo negativo de la probabilidad asignada al ítem que se debe predecir.

Sin embargo, a diferencia de modelos más simples como la regresión lineal, la obtención de los parámetros $\mathbf{W}^{(l)}$ y $\boldsymbol{b}^{(l)}$ no admite una solución explícita, pues el problema es no convexo. De esta manera, se debe recurrir a métodos de optimización iterativos para encontrar mínimos locales. Estos métodos calculan el gradiente de $\mathcal{L}$ con respecto a cada peso para saber cómo cambia el error ante pequeñas modificaciones en los parámetros. El cálculo se realiza mediante el algoritmo de retropropagación (*backpropagation*), el cual aplica de forma sistemática la regla de la cadena desde la capa de

salida hacia atrás, actualizando los pesos paso a paso en la dirección que reduce el error. El método clásico de optimización para obtener los parámetros es el *descenso del gradiente*, aunque en este caso se utilizará una variante moderna más eficiente: `AdamW` (puede consultarse en Loshchilov y Hutter, 2019).

Una vez entrenada, la red neuronal se puede emplear para la inferencia. Dados unos datos de entrada completamente nuevos (no conocidos durante el entrenamiento), la información se propaga a través de la red con los pesos ya determinados, llegando a una predicción final en la capa de salida.

Aprendizaje profundo

Hasta este punto, se ha descrito el funcionamiento de una red neuronal clásica. Estas arquitecturas resultan insuficientes cuando se enfrentan a datos lo suficientemente complejos. Para esta tarea de recomendación, y para conseguir una mejora predictiva con respecto a modelos clásicos (véase la Sección 3.1), se requieren modelos con una capacidad de abstracción mucho mayor. Es aquí donde entra el aprendizaje profundo (o *deep learning*). El término surge para denominar a redes neuronales que emplean múltiples capas ocultas. Esto permite una mayor capacidad para describir los datos, a cambio de un incremento en el coste computacional. De ahí que la mayor parte de redes neuronales *profundas* entrenadas a día de hoy utilicen unidades de procesamiento gráfico (GPU) muy potentes.

Para datos donde el orden es importante (como en *sequence modeling*), se desarrollaron numerosas arquitecturas de aprendizaje profundo a lo largo de las últimas décadas. Las redes recurrentes clásicas (RNN, véase Rumelhart et al., 1986) y las *long short-term memory* (LSTM, véase Hochreiter y Schmidhuber, 1997) nacieron para esta tarea. Estas últimas fueron las más usadas durante las dos siguientes décadas a su nacimiento, con aplicaciones como el reconocimiento de voz, la traducción automática o la predicción de series temporales. Sin embargo, sufrían severos problemas de memoria y paralelización (esencial para el tiempo de entrenamiento).

En 2017, ocho científicos de Google publicaron el artículo *Attention is All You Need* (véase Vaswani et al., 2017), presentando una nueva arquitectura de red neuronal profunda llamada Transformer. La novedad vino de sustituir el mecanismo de recurrencia de las redes anteriores por otro de *atención*, en el que toda la secuencia se procesa al mismo tiempo (haciéndolo paralelizable), y la propia red decide a qué *tokens* (en nuestro caso, ítems) prestarle más o menos atención. En la actualidad, las arquitecturas Transformer o derivadas son las más populares para una multitud de tareas, sobresaliendo en procesamiento de lenguaje natural y visión por computador (véanse *ChatGPT*, *Gemini* o *Claude*). Aún así, los Transformer encuentran varias limitaciones que se presentarán en el Capítulo 3, introduciendo nuevas arquitecturas como Mamba o xLSTM.

Mecanismos de regularización y estabilización

Entrenar redes neuronales profundas conlleva algunos desafíos, entre las cuales se encuentra el riesgo de sobreajuste (*overfitting*) y la inestabilidad numérica provocada por las fluctuaciones en escala y varianza de las activaciones. Por ello, se introducirán dos clases de capa que intentan mitigar estas problemáticas.

- **Capa Dropout:** Diseñada para combatir el sobreajuste, esta técnica desactiva aleatoriamente una fracción de las neuronas de una capa oculta en cada iteración del entrenamiento. Si denotamos por $p \in (0,1)$ a la probabilidad de desactivación, el *dropout* aplica una máscara estocástica $\boldsymbol{r}^{(l)}$ sobre las activaciones $\boldsymbol{y}^{(l)}$ antes de ser multiplicadas por sus pesos. Es decir,

$$\begin{aligned} \tilde{\boldsymbol{y}}^{(l)} &= \boldsymbol{y}^{(l)} \odot \boldsymbol{r}^{(l)}, \quad \text{con } r_i^{(l)} \sim \text{Bernoulli}(1-p), \\ \boldsymbol{y}^{(l+1)} &= \sigma\left(\mathbf{W}^{(l)}\tilde{\boldsymbol{y}}^{(l)} + \boldsymbol{b}^{(l)}\right), \end{aligned} \tag{1.6}$$

donde $\odot$ representa el producto elemento a elemento e $\tilde{\boldsymbol{y}}^{(l)}$ son las activaciones regularizadas. Esta capa fuerza a la red a distribuir el aprendizaje, eliminando posibles dependencias de algunas neuronas específicas. Durante la fase de inferencia, esto se elimina y todas las neuronas permanecen activas.

- **Capa LayerNorm:** A medida que los datos atraviesan una red, la distribución de las activaciones puede sufrir grandes cambios, lo que puede dificultar la convergencia del algoritmo de optimización. Para estabilizar la red se emplea una normalización de capa, ajustando la media y la varianza de las componentes del vector $\boldsymbol{z}^{(l)}$:

$$\begin{aligned} \mu^{(l)} &= \frac{1}{d}\sum_{i=1}^{d} z_i^{(l)}, \quad \sigma_z^2 = \frac{1}{d}\sum_{i=1}^{d}\left(z_i^{(l)} - \mu^{(l)}\right)^2, \\ \tilde{\boldsymbol{z}}^{(l)} &= \frac{\boldsymbol{z}^{(l)} - \mu^{(l)}}{\sqrt{\sigma_z^2 + \epsilon}} \odot \boldsymbol{\gamma}^{(l)} + \boldsymbol{\beta}^{(l)}, \\ \boldsymbol{y}^{(l+1)} &= \sigma\left(\tilde{\boldsymbol{z}}^{(l)}\right), \end{aligned} \tag{1.7}$$

  donde $\boldsymbol{\gamma}^{(l)}, \boldsymbol{\beta}^{(l)} \in \mathbb{R}^d$ son dos vectores de parámetros afines aprendibles propios de esta capa, y ϵ es una constante infinitesimal para evitar la división por cero.

1.3. Objetivos

La realización del presente TFM se basó en tres pilares fundamentales: investigación, desarrollo y comparación. Estos serán discutidos de manera detallada por un lado en el Capítulo 3 (investigación y desarrollo) y por otro en los capítulos 4 y 5 (comparación).

Investigación

Se realizó un extenso estudio acerca de las principales metodologías que han sido usadas para tareas de recomendación a lo largo del tiempo. Esto incluye algoritmos clásicos de filtrado colaborativo y filtrado por contenido, redes neuronales profundas para recomendación secuencial y métodos modernos colaborativos y por contenido. En particular:

- **Algoritmos clásicos:** Se estudiaron los métodos colaborativos Item-KNN, User-KNN y SVD, que usan cierta matriz de interacciones usuario-ítem. También se consideró TF-IDF, método clásico de filtrado por contenido.
- **Redes neuronales profundas para recomendación secuencial:** Principalmente el Transformer, explorando alternativas modernas como Mamba y xLSTM. Estas intentan paliar algunas de las problemáticas que sufre el primero.
- **Métodos modernos:** Para el filtrado colaborativo, investigando diversas arquitecturas de redes neuronales de grafos (GNN), y para el filtrado por contenido, empleando redes neuronales pre-entrenadas con una cantidad de datos masiva: BERT.

Integración

El objetivo del TFM no es tan solo comparar estos modelos, sino combinarlos para crear arquitecturas más potentes y/o escalables, que logren superar a las "simples". Como se verá a continuación, la potencia de un modelo se medirá a partir de una serie de métricas convenientemente definidas. Por otro lado, la escalabilidad dependerá directamente de la complejidad del modelo (en términos

computacionales) y la magnitud de los datos.

La idea es obtener información valiosa proveniente tanto de las GNN como de modelos BERT para alimentar y enriquecer a las redes neuronales profundas (Transformer, Mamba, xLSTM). Para ello existen diversos métodos, como la concatenación o suma de vectores, que serán desarrollados en el Capítulo 3. Las nuevas arquitecturas resultantes de la fusión de varios modelos serán denominadas **modelos híbridos**.

Comparación

En el Capítulo 5, se pondrá a examen el rendimiento de cada arquitectura estudiada, para buscar aquellas áreas donde sobresalen o muestran debilidades. Para ello se hará uso de diversas métricas, que se definirán convenientemente en el Capítulo 4:

- **Hit Rate@K, MRR@K y NDCG@K:** Estas son métricas clásicas para tareas de recomendación. Miden qué tan preciso es el modelo a la hora de acertar la película objetivo.
- **Personalización:** Lo interesante es que un modelo se ajuste lo máximo posible a cada perfil de usuario, con lo que esto conlleva. Predecir siempre "taquillazos" podría garantizar métricas de precisión decentes, a cambio de tener un alto "sesgo de popularidad".
- **Tiempo de entrenamiento o de computación:** Representa el coste computacional de cada modelo. Un incremento marginal en la precisión del sistema podría no justificar una carga computacional excesiva, sobre todo en entornos de producción.

Capítulo 2

Datos

En el área de los sistemas de recomendación, existen varios *datasets* "clásicos". Se optó por emplear MovieLens (véase Harper y Konstan, 2015), probablemente el más utilizado para comparar y evaluar estos sistemas en el ámbito académico. MovieLens es un conjunto de datos sobre valoraciones de películas desarrollado por el equipo de investigación GroupLens, de la Universidad de Minnesota. Existen numerosas versiones del mismo dependiendo del tamaño. En particular, y por su recomendación en entornos de desarrollo, se ha escogido la versión *Full*, que contiene aproximadamente 33 millones de valoraciones de películas comprendidas entre 1 y 5 estrellas. Estas valoraciones se reparten en 330.975 usuarios, considerando una cantidad de 86.000 películas. Los datos se encuentran estructurados en tres tablas:

- **Películas (`movies.csv`):** contiene la lista de todas las películas registradas en MovieLens. La conforman tres columnas: `movieId` (identificador de película de MovieLens), `title` (título de película) y `genres` (géneros de cada película).
- **Valoraciones (`ratings.csv`):** contiene cada valoración que ha dado cada usuario de MovieLens. Columnas: `userId` (identificador de usuario), `movieId` (identificador de película de MovieLens), `rating` (valoración de película por usuario, en una escala de 1.0 a 5.0 con incrementos de 0.5) y `timestamp` (día y hora de la valoración, en formato Unix).
- **Nexos (`links.csv`):** permite enlazar películas de Movielens (columna `movieId`) con metadatos procedentes de IMDb (columna `imdbId`) o TheMovieDatabase (columna `tmdbId`). Véanse respectivamente IMDb.com, Inc. (2026) y The Movie Database (2026).

2.1. Naturaleza de los datos

Emplear MovieLens tiene un gran punto positivo: al ser el más popular, es sencillo encontrar modelos ya evaluados en el mencionado conjunto de datos. Además, en el Capítulo 4 se presentan algunas de las métricas más utilizadas, lo que facilita la comparación directa de los modelos obtenidos con los resultados publicados en otros artículos. Por otro lado, es un *dataset* bien estructurado, que no requiere un gran esfuerzo en cuestión de limpieza de datos. Es de uso libre, y conviene destacar que los usuarios son reales, es decir, no se trata de datos sintéticos.

Sin embargo, no todo es perfecto. Fan et al. (2024) mostraron una posible debilidad del *dataset*. Los buenos resultados de un modelo sobre MovieLens en cuestión de métricas no siempre se trasladan a la práctica. Esto sucede por dos motivos:

- La manera en la que los usuarios valoran películas, es decir, la forma en la que la secuencia de ítems es generada, está altamente influenciada por el algoritmo de recomendación interno de la propia plataforma.

- Además, las primeras valoraciones de un usuario (los primeros ítems de cada secuencia) presentan una variabilidad muy baja, y definen en gran parte las próximas películas que le serán ofrecidas al usuario para valorar.

Aunque quizás no debiera ser la única justificación para demostrar la efectividad de un sistema de recomendación, el *dataset* de Movielens sigue siendo de gran utilidad. Para acercarse algo más a un caso real (por ejemplo, implementar un modelo de recomendación para una plataforma de *streaming*), los autores recomiendan truncar las secuencias por la izquierda. En particular, en este trabajo se ha decidido eliminar las 20 primeras películas de cada usuario. Como se puede ver en la Figura 2.1, este "sesgo" se reduce considerablemente, sobre todo a medida que se aumenta la cantidad de películas truncadas. La decisión de tomar 20 como referencia responde a dos cuestiones. Por un lado, efectuar un truncamiento más ligero no supondría una diferencia resaltable con el conjunto original. Por otro lado, un truncado más agresivo uniformizaría aún más la distribución de popularidad, suponiendo sin embargo una pérdida drástica de información y una manipulación excesiva de los historiales de usuario.

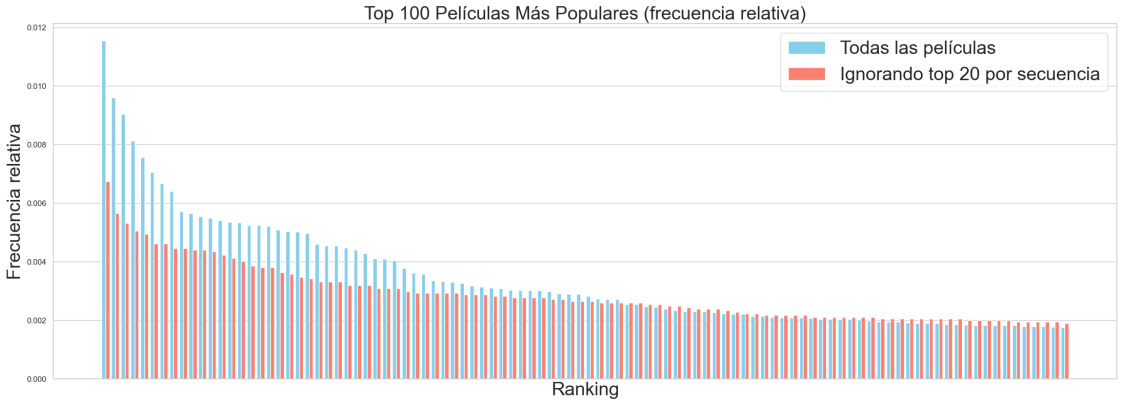

Figura 2.1: Frecuencia relativa de las 100 películas más populares, considerando las secuencias completas (azul) y truncadas (rojo).

Estas secuencias truncadas se utilizarán de forma paralela a las originales para comparar el rendimiento de las arquitecturas. De esta forma, se contrastará el rendimiento de las arquitecturas bajo dos condiciones diferenciadas, determinando en qué medida la capacidad predictiva de cada modelo puede depender de sesgos de correlación iniciales.

2.2. Generación de secuencias

El uso de redes neuronales profundas sobre grandes volúmenes de datos (como la versión *Full* de MovieLens) exige un coste computacional sumamente elevado, tanto en la demanda de memoria (VRAM) como en los tiempos de entrenamiento. Dado que para el desarrollo de este trabajo se ha dispuesto de una tarjeta gráfica NVIDIA Tesla T4, una de las más básicas para computación en la nube, procesar el conjunto de datos en su totalidad resulta inviable. De esta manera, es crucial reducir el tamaño del dataset para adecuarlo a las herramientas disponibles. Por ello, se ha seguido una estrategia de preprocesamiento mediante ciertos parámetros que se comentarán a continuación (`MIN_LENGTH`, `MAX_LENGTH`, `RANDOM_SAMPLE` y `MIN_RATING`). Este se llevó a cabo en el siguiente orden:

1. **Generación:** teniendo en cuenta la fecha y hora de visualización (`timestamp`), se generan secuencias ordenadas conformadas por todas las películas que cada usuario ha visto.

2. **Truncado por la izquierda:** Como se mencionaba anteriormente, cada modelo será entrenado de dos maneras. Por un lado, con los datos "originales" (omitiendo este paso) y por otro, a partir de las secuencias truncadas por la izquierda. De esta manera, generaremos dos *inputs* distintos.

3. **Truncado por longitud de secuencia:** Aquí entran los dos parámetros `MIN_LENGTH` y `MAX_LENGTH`, que restringen a cada usuario según cuántas películas hayan visto. En particular, se usará un intervalo de 5 a 90 ítems. Los usuarios con pocos ítems pueden considerarse inactivos y carecen de suficiente relevancia (representan aproximadamente un 7.3 % del total). Por otro lado, un historial más largo proporcionará en general una mejor idea del contexto de usuario. Sin embargo, y como se verá en el Capítulo 4, ciertos modelos sufren especialmente al aumentar el parámetro `MAX_LENGTH`. Por ello se establecerá el límite en 90, eliminando así a un 26.1 % de los usuarios (véase la Figura 2.2).

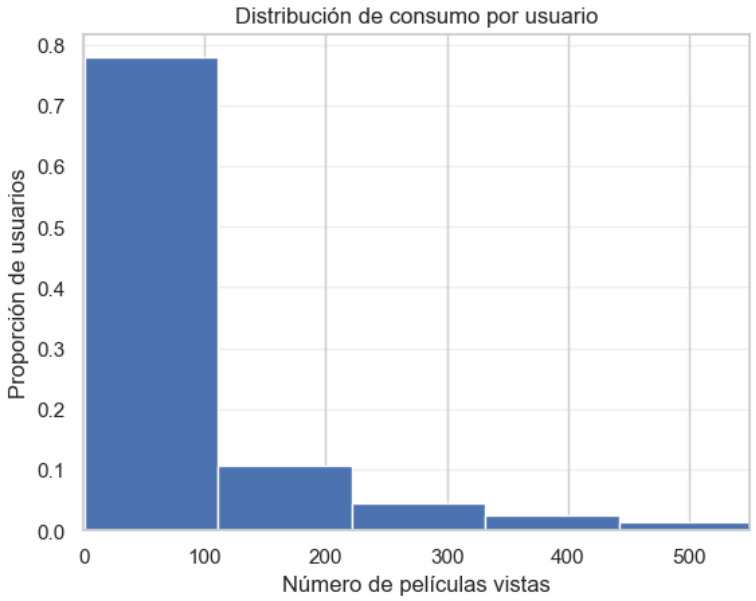

Figura 2.2: Distribución del número de películas vistas por usuario.

4. **Muestreo aleatorio simple:** Tan solo se toma una porción de los usuarios totales, usando la función `random.sample()`. En este caso, se empleó `RANDOM_SAMPLE` = 0.3. El porqué de esta decisión tiene que ver con el tiempo de entrenamiento. Se observó que una fracción cercana al 30 % no pierde demasiado en rendimiento (métricas de precisión), en comparación a emplear el conjunto íntegro de datos, incluso mejorando hasta tres veces en términos de velocidad. Por otro lado, disminuirlo más allá del 0.3 otorga resultados significativamente peores. Aunque el significado de la siguiente tabla se verá en profundidad en el Capítulo 4, es posible apreciar este tema en la Tabla 2.1.

5. **Filtro de calidad:** Finalmente, se eliminan aquellas películas cuyas valoraciones se hallen por debajo de un filtro de calidad `MIN_RATING=3.0`. Por ejemplo, si el usuario 1 ha valorado la película *Sharknado* con un 1.0, esta será ignorada para la secuencia final de usuario. Esto se hace porque se desarrollarán modelos "ciegos" a las valoraciones: la intención (como se verá en la Sección 3.2) es que el modelo capture la secuencialidad de un simple historial cronológico para ofrecer su recomendación. Añadir las valoraciones de usuario como contexto podría ser una buena manera de obtener mejores resultados (permitiendo a su vez eliminar este filtro de calidad), mas no es el objetivo principal de este TFM.

De esta manera, para la versión con datos sin truncar se obtienen 12839 usuarios, 13318 películas y 755591 valoraciones. Para las arquitecturas secuenciales se ha seguido la estrategia tradicional de partición: 80 % para datos de entrenamiento, 10 % para validación y 10 % para test. Así, resultan

RANDOM_SAMPLE	HR@1	HR@20	HR@300	MRR@300	Usuarios	Tiempo (min)
0.03	0.00	7.69	47.3	0.0265	909	0.95
0.1	4.28	19.7	64.5	0.0740	3039	2.31
0.3	5.91	26.2	72.6	0.1077	9121	5.54
0.5	6.98	28.9	73.4	0.1215	15163	9.47
1	6.09	28.6	76.8	0.1133	30364	18.41

Tabla 2.1: Resultados de métricas y tiempos de ejecución dependiendo del tamaño del dataset. Modelo Transformer, entrenado con las secuencias truncadas. La primera columna representa el porcentaje de usuarios utilizados sobre el total de MovieLens. Las columnas 2 a 5 son los valores de distintas métricas que se definirán en el Capítulo 4. Valores mayores se asocian con mejores rendimientos. La sexta columna se corresponde con el número de usuarios finales en el conjunto de entrenamiento. Por último, la séptima columna indica el tiempo precisado para entrenar la red neuronal.

10271 secuencias disponibles para entrenar cada modelo. Véase la naturaleza de las mismas en la Figura 2.3. Por otro lado, para la versión con datos truncados se obtendrían 9121 secuencias para entrenamiento. Esta diferencia de tamaño del 12 % (debida a la manera en que se generan las secuencias) no es preocupante, y no supondrá una justificación suficiente para la disparidad de resultados que se observarán en el Capítulo 5.

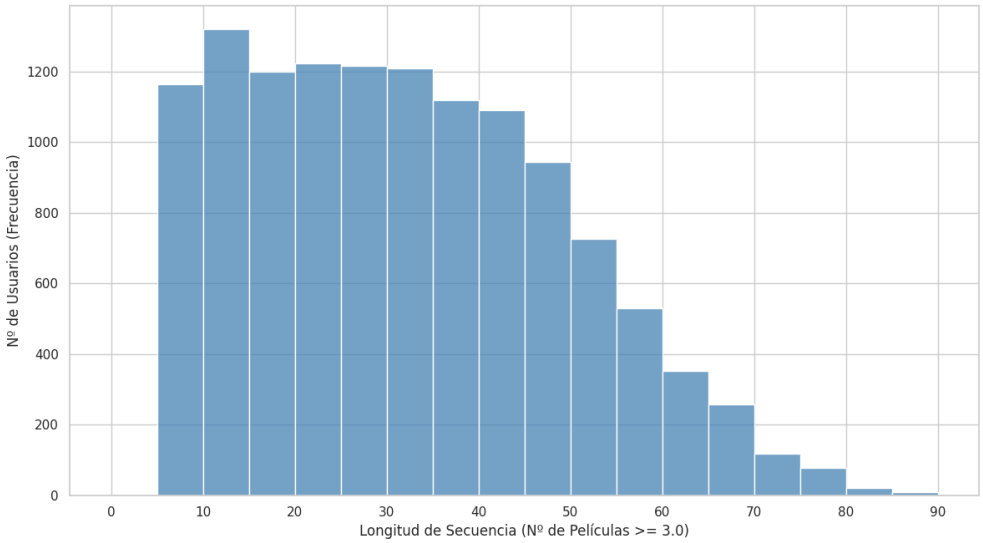

Figura 2.3: Distribución de la secuencias finales en función de su longitud (datos originales), después de truncar por longitud de secuencia y de filtrar las valoraciones menores a 3 estrellas.

Capítulo 3

Arquitecturas

El campo de los sistemas de recomendación ha ido evolucionando a lo largo de las últimas décadas (véase la Sección 1.1): desde enfoques estadísticos estáticos hasta modelos de aprendizaje profundo, capaces de capturar patrones de comportamiento mucho más complejos. El propósito de este capítulo es describir formalmente los modelos trabajados, comenzando por los métodos clásicos de filtrado colaborativo, continuando con las redes neuronales modernas para *sequence modeling* y describiendo una serie de implementaciones híbridas que aprovechan la posibilidad de combinar modelos. El capítulo se estructurará de la siguiente forma:

- **Métodos clásicos:** Se analizará el algoritmo **Item-KNN**, el más potente entre los tradicionales del filtrado colaborativo. Destaca por su interpretabilidad y capacidad para establecer similitudes directas entre ítems basándose tan solo en matrices de interacción.
- **Modelos secuenciales modernos:** Se profundizará en la arquitectura **Transformer**, que revolucionó el procesamiento de secuencias mediante un mecanismo de atención, permitiendo capturar dependencias globales en el historial del usuario. Además, se analizarán los modelos **Mamba** y **xLSTM**, nacidos para solventar algunas de las problemáticas presentes en el Transformer.
- **Arquitecturas híbridas:** Por último, se abordará la integración de estos modelos secuenciales modernos con técnicas complementarias, como las **redes neuronales de grafos (GNN)** y modelos de lenguaje pre-entrenados como **BERT**. El objetivo es desarrollar arquitecturas más potentes, enriqueciendo las representaciones secuenciales con información colaborativa y semántica adicional para maximizar la precisión en la predicción del NBI.

3.1. Item-KNN

El método de los K vecinos más cercanos se enmarca dentro de los modelos de filtrado colaborativo. Fue popularizado por Sarwar et al. (2001) y se ha consolidado como uno de los algoritmos más básicos para la recomendación de ítems. Es un método estático, pero que igualmente precisa un historial para cada usuario. En particular, hace uso de una matriz de interacciones $\mathbf{M} \in \mathbb{N}^{U \times I}$, donde U es el número de usuarios, I el número de ítems y cada elemento $m_{ui} \in \mathbf{M}$ se obtiene de la siguiente forma:

$$m_{ui} = \begin{cases} 1 & \text{si el usuario } u \text{ ha consumido el ítem } i, \\ 0 & \text{en otro caso.} \end{cases}$$

Una vez obtenida esta matriz, que puede ser de gran tamaño, surgen diversas algoritmias dentro del filtrado colaborativo con la finalidad última de componer una recomendación lo más personalizada posible para cada usuario. Entre ellas se hallan la descomposición en valores singulares (SVD), el User-KNN y el Item-KNN. Debido a su mejor desempeño en el contexto de MovieLens, y su alta

similitud con los otros dos métodos (en construcción), se ha decidido centrar el análisis en este último. En el Capítulo 5 se comentarán los resultados obtenidos con estos modelos sobre los datos.

Veamos en detalle cómo se construyen las recomendaciones para el Item-KNN.

Método 3.1 (*Item-KNN*)**.**

1. *En primer lugar, se obtiene la matriz de interacciones* $\mathbf{M} \in \mathbb{N}^{U \times I}$*. Cada ítem* i *es representado por su vector columna* $\boldsymbol{c}_i$ *correspondiente, que representa el conjunto de usuarios que ha interactuado con él.*

2. *Seguidamente, se calcula una matriz de similitudes* $\mathbf{S} \in \mathbb{R}^{I \times I}$*, donde cada elemento* s_{ij} *representa la similitud colaborativa entre dos ítems. Definimos la similitud coseno como sigue*

$$sim(i,j) = \frac{\boldsymbol{c}_i^\top \cdot \boldsymbol{c}_j}{||\boldsymbol{c}_i||_2 \cdot ||\boldsymbol{c}_j||_2}, \tag{3.1}$$

   siendo $\boldsymbol{c}_i, \boldsymbol{c}_j$ *los vectores columna de* $\mathbf{M}$ *correspondientes a los ítems* i, j*, y* $||\cdot||_2$ *la norma euclidiana. Esto resulta en una matriz simétrica, cuyas filas (o columnas) indican las puntuaciones de similitud de un ítem determinado con el resto.*

3. *Para la inferencia en un usuario, se consideran todos sus ítems (o los últimos* n *consumidos) y se agregan las columnas de la matriz de similitud* $\mathbf{S}$ *correspondientes. El vector resultante reflejará una puntuación para cada ítem. Enmascarando los ítems ya vistos, y extrayendo las* K *puntuaciones más altas, se obtiene la recomendación final.*

Ejemplo 3.2.

1. Considérese la matriz de interacciones

$$\mathbf{M} = \begin{array}{c} \\ u_1 \\ u_2 \\ u_3 \\ u_4 \end{array} \begin{array}{c} \begin{array}{ccc} i_1 & i_2 & i_3 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix} \end{array},$$

   donde el usuario u_1 ha visto el ítem i_2, el usuario u_2 ha visto los ítems i_1 e i_2, etc.

2. Calculemos la matriz de similitudes entre ítems:

$$\mathbf{S} = \begin{pmatrix} 1 & 0.82 & 0.5 \\ 0.82 & 1 & 0.41 \\ 0.5 & 0.41 & 1 \end{pmatrix}.$$

3. Veamos las recomendaciones para los usuarios u_2 y u_3, agregando las columnas en $\mathbf{S}$ correspondientes a los ítems que ha consumido. Entonces, como el usuario u_2 ha consumido los ítems i_1, i_2, y el usuario u_3 ha consumido tan solo el ítem i_3:

   - u_2: $(1.00, 0.82, 0.50) + (0.82, 1.00, 0.41) = (1.82, 1.82, 0.91)$,
   - u_3: $(0.50, 0.41, 1.00)$,

   y enmascarando los ítems ya vistos (poniéndolos a cero):

   - u_2: $(0.00, 0.00, 0.91)$.
   - u_3: $(0.50, 0.41, 0.00)$.

Así, para $K = 1$, la recomendación para el usuario u_2 es el ítem i_3, y para el usuario u_3 es el ítem i_1, seguido por el ítem i_2.

El Item-KNN pertenece a los métodos de *lazy learning*, donde la inferencia es "en el momento" (no es posible entrenar el modelo como sí hacen las redes neuronales). Sin embargo, la gran ventaja de este método es precisamente su velocidad. Para conjuntos de datos similares al nuestro, la matriz de interacciones $\mathbf{M}$ suele ser extremadamente dispersa, ya que la inmensa mayoría de los usuarios solo ha consumido una fracción mínima del catálogo total de películas. Aprovechando esta dispersión computacionalmente, el cálculo de la matriz de similitudes $\mathbf{S}$ requiere muy pocos recursos en comparación con el entrenamiento e inferencia de una red neuronal profunda. Por estas razones, el Item-KNN se establece como un excelente modelo base (*baseline*) con el que contrastar el rendimiento de arquitecturas más modernas y complejas.

3.2. Transformer

Como adelantábamos en el Capítulo 1, la arquitectura Transformer fue introducida en el año 2017 por la multinacional Google con el artículo *Attention is all you need* (véase Vaswani et al., 2017), convirtiéndose en uno de las más influyentes de la literatura científica, y superando las 200 mil citas según Google Scholar. El Transformer es una arquitectura de red neuronal profunda, cuyo propósito incicial era el de mejorar técnicas en el campo del procesamiento de lenguaje natural (*NLP*), como la traducción automática. Aún así, han surgido incontables aplicaciones: visión por computador (véase Dosovitskiy et al., 2021), modelado de series temporales (véase Lim et al., 2021), generación de texto (véase Radford et al., 2019), etc.

Como se puede observar en la Figura 3.1, la arquitectura original consta de dos grandes bloques: el *encoder* (codificador) y el *decoder* (decodificador). El primero fue diseñado para procesar la secuencia de entrada, extrayendo el contexto y construyendo una representación profunda de los datos. Por otro lado, el *decoder* se concibió para tareas de traducción secuencia a secuencia (*seq2seq*, puede consultarse en Sutskever et al., 2014), generando un nuevo texto paso a paso mediante un mecanismo de atención cruzada (*cross-attention*) que consulta continuamente la información proveniente del *encoder*. En este caso, el objetivo es comprender a fondo el historial cronológico del usuario para realizar una única tarea de clasificación multiclase: estimar la probabilidad del próximo ítem a consumir. Por este motivo resulta innecesario emplear la arquitectura original por completo. Así, se opta por usar una pila de bloques *encoder*. Estos bloques poseen la capacidad de abstracción suficiente para captar las dependencias a corto y largo plazo entre las películas del historial.

El *encoder* original es bidireccional (es decir, procesa toda la secuencia a la vez). Sin embargo, para este problema de recomendación veremos que se debe inyectar una "máscara" causal dentro del mecanismo de atención. Esta máscara causal no es más que un manera de cegar al modelo, evitando que pueda mirar al futuro en cada paso de tiempo t. De esta forma evitaremos la fuga de datos durante el entrenamiento, garantizando que la representación de una película en el instante t se construya basándose única y exclusivamente en las películas vistas con anterioridad.

3.2.1. Modelado

El objetivo es transformar la secuencia de interacciones del usuario en una estructura que el Transformer pueda procesar. Sea $\mathcal{I} = \{1, 2, \ldots, |\mathcal{I}|\}$ el conjunto de ítems únicos (películas) y u un usuario con un historial cronológico de n interacciones $S^u = (s_1, s_2, \ldots, s_n)$, donde cada $s_t \in \mathcal{I}$. Estos s_t serán codificados en tres pasos: codificación de película, codificación de posición y agregación de ambas.

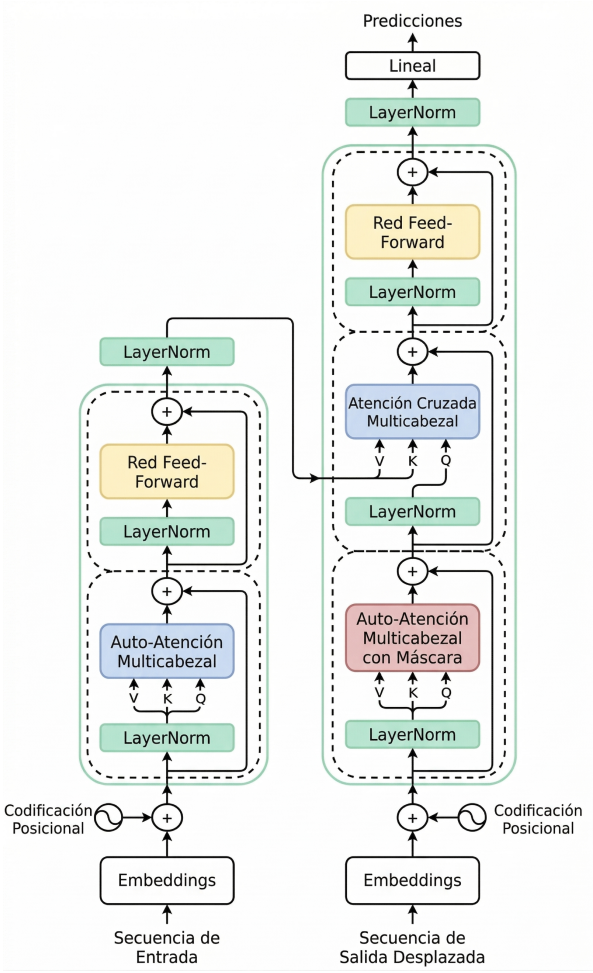

Figura 3.1: Esquema de la arquitectura Transformer. A la izquierda, el bloque *Encoder*, junto con los mecanismos de atención multicabezal y red *feed-forward*. A la derecha, el bloque *Decoder*. Edición de la imagen original en Vaswani et al. (2017).

1. En primer lugar, cada identificador de ítem $i \in \mathcal{I}$ se mapea a un espacio de dimensión d (por ejemplo, $d = 512$). Esta representación vectorial de cada ítem se denomina *embedding*. Así, se define la matriz de *embeddings* de ítems como $\mathbf{M} \in \mathbb{R}^{|\mathcal{I}| \times d}$. Esta matriz $\mathbf{M}$ se actualizará durante el entrenamiento. Denominamos $\boldsymbol{m_i}$ a la fila correspondiente al ítem i.

2. Por otro lado, y dado que en un principio el Transformer no posee una noción del orden de los elementos (pues no hay recurrencia), se utiliza codificación posicional, o *positional encoding*. Para ello, se hace uso de una matriz de *embeddings* de posición $\mathbf{P} \in \mathbb{R}^{N \times d}$, donde N es la longitud máxima de la secuencia (por ejemplo $N = 90$). Para un ítem i en una posición determinada $t \in \{0, \ldots, N-1\}$, las componentes del vector de posición $\boldsymbol{p}_t \in \mathbb{R}^d$ se calculan de la siguiente manera:

$$\boldsymbol{p}_t^j = \begin{cases} \sin\left(\dfrac{t}{10000^{j/d}}\right), & \text{si } j \text{ es par,} \\ \cos\left(\dfrac{t}{10000^{(j-1)/d}}\right), & \text{si } j \text{ es impar,} \end{cases}$$

donde $j \in \{0, \ldots, d/2 - 1\}$. Según Vaswani et al. (2017), esta formulación permite al modelo entender la estructura cronológica fácilmente, pues $\boldsymbol{p}_{t+k}$ se puede representar como una función lineal de $\boldsymbol{p}_t$ para todo $k < N$. La fórmula utiliza potencias de 10000 para conseguir representar grandes longitudes de secuencia, distinguiendo relaciones locales (t bajo, alta frecuencia de onda) y globales (t alto, baja frecuencia).

3. La representación final del ítem i en la posición t de la secuencia de usuario S^u, $\boldsymbol{h}_t^0$, se obtiene mediante la suma de ambos *embeddings*, aplicando posteriormente una capa de *dropout* (véase la Ecuación 1.6) para evitar el sobreajuste:

$$\boldsymbol{h}_t^0 = \text{Dropout}(\boldsymbol{m}_i \cdot \sqrt{d} + \boldsymbol{p}_t).$$

Agrupando los vectores individuales de cada ítem s_t en la secuencia se construye la matriz de entrada $\mathbf{H}^0 \in \mathbb{R}^{N \times d}$. Notar que N es tan solo la longitud máxima: así, una secuencia de $n < N$ interacciones sencillamente se completaría con filas vacías. Computacionalmente, esto se lleva a cabo con una máscara denominada **PAD**, que habrá que considerar además de la ya mencionada máscara causal.

El modelo empleado consiste en una pila de L bloques *encoder* idénticos (por ejemplo, $L = 2$). Cada bloque toma la representación de la capa anterior $\mathbf{H}^{(l-1)} \in \mathbb{R}^{N \times d}$ y genera una nueva representación contextualizada $\mathbf{H}^{(l)}$. Cada uno de estos bloques se compone de dos subcapas principales: el mecanismo de atención multicabezal (*multi-head self-attention*) y la red *feed-forward*.

Mecanismo de atención multicabezal

Esta capa permite al modelo capturar dependencias entre los ítems de la secuencia de un usuario. Para ello se utilizan múltiples "cabezales de atención", permitiendo que el modelo atienda a diferentes características o patrones de forma separada. Por ejemplo, un cabezal podría especializarse en detectar relaciones basadas en el género cinematográfico, mientras que otro podría enfocarse en la proximidad temporal de una valoración con respecto a otra. Para cada cabezal de atención $h \in \{1, \ldots, \texttt{N\_HEAD}\}$ (por ejemplo, $\texttt{N\_HEAD} = 8$), se proyecta la matriz de entrada $\mathbf{H}^{(l-1)}$ a través de tres matrices de pesos entrenables para obtener las matrices de *consultas* ($\mathbf{Q}_h$), *claves* ($\mathbf{K}_h$) y *valores* ($\mathbf{V}_h$):

$$\begin{aligned}
\mathbf{Q}_h &= \mathbf{H}^{l-1}\mathbf{W}_h^Q \quad \in \mathbb{R}^{N \times d_k}, \\
\mathbf{K}_h &= \mathbf{H}^{l-1}\mathbf{W}_h^K \quad \in \mathbb{R}^{N \times d_k}, \\
\mathbf{V}_h &= \mathbf{H}^{l-1}\mathbf{W}_h^V \quad \in \mathbb{R}^{N \times d_v},
\end{aligned}$$

donde $\mathbf{W}_h^Q, \mathbf{W}_h^K \in \mathbb{R}^{d \times d_k}$ y $\mathbf{W}_h^V \in \mathbb{R}^{d \times d_v}$ (por ejemplo, $d_k = d_v = d/\texttt{N\_HEAD}$). La *consulta* de cada ítem (filas de $\mathbf{Q}_h$) indica lo que el ítem busca en la secuencia para entender su propio contexto. Además, la *clave* representa cómo se describe cada ítem. Por último, el *valor* se puede interpretar como la representación del contenido "real" del ítem.

La salida de cada cabezal se calcula de la siguiente manera:

$$\text{head}_h = \text{SoftMax}\left(\frac{\mathbf{Q}_h\mathbf{K}_h^\top}{\sqrt{d_k}} + \mathbf{MASK}\right)\mathbf{V}_h,$$

siendo **MASK** una máscara causal que evita que el modelo "mire hacia el futuro" durante el entrenamiento. Se representa como una matriz triangular superior donde los elementos por encima de la diagonal principal tienden a $-\infty$:

$$\mathbf{MASK}_{ij} = \begin{cases} 0, & \text{si } i \geq j, \\ -\infty, & \text{si } i < j. \end{cases} \tag{3.2}$$

El modelo mide la similitud entre consultas y claves, pasándolos por una capa SoftMax para elegir a qué prestarle más o menos atención. Por ejemplo, al calcular los pesos de atención de la película *Shrek II* dentro la siguiente secuencia

$$(\ \textit{Shrek},\quad \textit{Love Actually},\quad \textit{Shrek II},\quad \textit{Shrek Tercero}\ ),$$

podríamos obtener unos pesos de este estilo:

$$(\ 0.8,\quad 0.2,\quad \bullet,\quad 0\ ).$$

Finalmente, las salidas de todos los cabezales se concatenan y se proyectan de nuevo a dimensión $d \times d$ para "mezclar" la información de cada cabezal de atención:

$$\text{MHA}(\mathbf{H}^{(l-1)}) = \text{Concat}(\text{head}_1, \ldots, \text{head}_{\texttt{N\_HEAD}})\mathbf{W}^O,$$

con $\mathbf{W}^O \in \mathbb{R}^{d\times d}$ una matriz de proyección con pesos aprendibles.

Red *feed-forward*

Esta subcapa se encarga de procesar individualmente cada ítem, una vez que ya se ha recolectado información a través del mecanismo de atención. En primer lugar, la salida del mecanismo de atención se suma a la entrada original del bloque $\mathbf{H}^{l-1}$. Así pues, el estado intermedio $\hat{\mathbf{H}}^l$ se define como:

$$\hat{\mathbf{H}}^{(l)} = \text{LayerNorm}(\mathbf{H}^{(l-1)} + \text{Dropout}(\text{MHA}(\mathbf{H}^{l-1}))),$$

donde se ha aplicado una capa de *dropout* (véase la Ecuación 1.6) y otra de normalización (véase la Ecuación 1.7). Sobre este estado intermedio se aplica una pequeña red neuronal, la red FF (*feed-forward*). Esta introduce la no-linealidad necesaria para que se puedan modelar interacciones complejas:

$$\text{FF}(\hat{\mathbf{H}}^{(l)}) = \text{ReLU}(\hat{\mathbf{H}}^{(l)}\mathbf{W}_1 + \boldsymbol{b}_1)\mathbf{W}_2 + \boldsymbol{b}_2, \tag{3.3}$$

donde $\mathbf{W}_1 \in \mathbb{R}^{d\times d_{ff}}$ y $\mathbf{W}_2 \in \mathbb{R}^{d_{ff}\times d}$ son matrices de pesos entrenables (por ejemplo, $d_{ff} = 4d$), y ReLU la función de activación definida en la Ecuación 1.2. Finalmente, se suma la entrada de la FF con su salida, seguida de una normalización final para obtener la representación de la capa l:

$$\mathbf{H}^{(l)} = \text{LayerNorm}(\hat{\mathbf{H}}^{(l)} + \text{Dropout}(\text{FF}(\hat{\mathbf{H}}^{(l)})) \tag{3.4}$$

Estos dos procesos (atención multicabezal y red FF) se repiten para cada una de las L capas del *encoder*, permitiendo que el modelo refine progresivamente la representación de cada película en una secuencia en función de su contexto temporal y semántico.

Capa de salida y predicción

La salida del último bloque, $\mathbf{H}^{(L)}$, contiene las representaciones finales de cada ítem en la secuencia de usuario. Para la tarea de recomendación, se toma el vector correspondiente a la última posición de la secuencia (embebido con la información de toda la cronología), $\boldsymbol{h}_N^{(L)} \in \mathbb{R}^d$, y se proyecta mediante una capa lineal al tamaño del vocabulario $|\mathcal{I}|$:

$$\hat{\boldsymbol{y}} = \boldsymbol{h}_N^{(L)}\mathbf{W}_{out} + \boldsymbol{b}_{out}, \tag{3.5}$$

con $\mathbf{W}_{out} \in \mathbb{R}^{d\times|I|}$ y $\boldsymbol{b}_{out} \in \mathbb{R}^{|I|}$ los pesos aprendibles. El vector resultante $\hat{\boldsymbol{y}} \in \mathbb{R}^{|\mathcal{I}|}$ contiene las puntuaciones brutas para cada ítem. Aplicando una última capa SoftMax (véase Ecuación 1.4) es posible transformar estas puntuaciones en probabilidades, restando una lista del catálogo de películas en la que cada una tiene una probabilidad asignada.

Al entrenar el modelo como veremos a continuación, se está realizando una estimación por máxima verosimilitud (véase Goodfellow et al., 2016). Así, la salida de la capa SoftMax converge a una distribución de probabilidad, donde cada componente de la predicción actúa como un estimador $\hat{p}_i$ de la probabilidad de que el usuario consuma dicho ítem, condicionado a su historial:

$$\hat{y}_i = \hat{p}_i \approx \mathbb{P}(s_{t+1} = i \mid S^u_{1:t}). \tag{3.6}$$

3.2.2. Entrenamiento

El modelo se entrena minimizando la función de pérdida $\mathcal{L}_{\text{CE}}$ (véase *cross-entropy loss*, Ecuación 1.5). Esta compara la predicción $\hat{\boldsymbol{y}}$ con el siguiente ítem real $\boldsymbol{y}$ (el NBI) que el usuario consumió en el conjunto de datos de MovieLens, para cada posición en la secuencia. Es decir, considerando una secuencia de $n < N$ interacciones $S^u = (s_1, s_2, \ldots, s_n)$, se computará $n-1$ veces la función de pérdida:

$$\begin{aligned}(s_1) &\longrightarrow s_2,\\ (s_1, s_2) &\longrightarrow s_3,\\ &\cdots\\ (s_1, s_2, \ldots, s_{n-1}) &\longrightarrow s_n.\end{aligned}$$

Para una posición t en la secuencia, si el ítem real consumido a continuación es $\boldsymbol{y}$ (en la posición s_{t+1}), la pérdida $\mathcal{L}_{\text{CE}}$ se define como el logaritmo negativo de la probabilidad asignada por el modelo al ítem correcto:

$$\mathcal{L}_{\text{CE}} = -\log\left(\hat{\mathbb{P}}(\boldsymbol{y}|S^u_{1:t})\right) = -\log(\hat{y}_{\boldsymbol{y}}), \tag{3.7}$$

donde $S^u_{1:t}$ denota la secuencia S^u hasta el instante t, e $\hat{y}_{\boldsymbol{y}}$ denota la probabilidad otorgada por la predicción $\hat{\boldsymbol{y}}$ al ítem $\boldsymbol{y}$. La penalización logarítmica implica que si el modelo asigna una probabilidad cercana a 0 al ítem real, la pérdida tiende a infinito, lo cual fuerza a los pesos a un ajuste más drástico. La pérdida total del modelo para una secuencia completa de longitud N es el promedio de las pérdidas individuales en cada paso de tiempo:

$$\mathcal{L}_{\text{total}} = \frac{1}{n}\sum_{t=1}^{n}\mathcal{L}_{\text{CE}}(t). \tag{3.8}$$

Es decir, la pérdida al predecir s_2 sabiendo $\{s_1\}$, más la pérdida de predecir s_3 conociendo $\{s_1, s_2\}$, y así sucesivamente. Gracias al uso de la máscara causal **MASK**, el Transformer puede paralelizar estas n operaciones durante el entrenamiento, optimizando considerablemente la eficiencia computacional.

3.2.3. Configuración de hiperparámetros

Gran parte de la eficacia del Transformer depende de una buena configuración de sus hiperparámetros, pues estos pueden influir en aspectos como la capacidad de representación (por ejemplo, la dimensión de los *embeddings* d) o la eficiencia del entrenamiento (por ejemplo, el *dropout*). Tras una fase de experimentación, se han seleccionado los valores detallados en la Tabla 3.1.

Hiperparámetro	**Valor**	**Descripción**
`D_MODEL` (d)	512	Dimensión del espacio de *embedding*.
`N_HEADS` (h)	8	Número de cabezales de atención.
`NUM_LAYERS` (L)	2	Bloques *encoder* apilados.
`D_FF` (d_{ff})	2048	Dimensión interna de la red FF.
`DROPOUT`	0.1	Tasa de regularización por abandono.

Tabla 3.1: Configuración de hiperparámetros para el modelo Transformer.

La elección de estos valores se fundamenta en los siguientes criterios:

- `D_MODEL`: Cuanto mayor sea la dimensión de los *embeddings*, mayor será su capacidad de representación. Se optó por tomar $d = 512$ que, como se puede observar en la Tabla 3.2, fue escogido en base a una fase de pruebas con diferentes valores. Tamaños menores consiguen peores resultados y, en ocasiones, mayores tiempos de entrenamiento. Por otro lado, tamaños excesivos pueden ocasionar problemas de asignación de memoria en la GPU (forzando a retocar otros parámetros).

Dim (d)	**HR@1**	**HR@20**	**HR@300**	**MRR@300**	**Tiempo (min)**
64	4.70	25.0	68.7	0.0933	11.80
128	7.25	29.0	73.5	0.1238	11.42
256	7.33	32.1	75.4	0.1337	9.83
512	9.09	32.9	75.1	0.1472	8.44
1024	7.81	33.7	75.8	0.1414	14.00

Tabla 3.2: Rendimiento del Transformer en función de la dimensión de los *embeddings* d. Versión con secuencias originales. Valores mayores para las columnas 2 a 5 se asocian con mejores rendimientos. La última columna indica el tiempo precisado para entrenar la red neuronal.

  Resulta llamativo observar que los tiempos de entrenamiento no crecen de forma monótona en función de la dimensión de los *embeddings*. Al tener una menor capacidad de representación, las arquitecturas más pequeñas necesitan entrenar durante un número mucho mayor de iteraciones, lo que a menudo puede significar tiempos mayores. Por otro lado, para tamaños excesivos ($d \geq 1024$), el coste computacional puede disparar los tiempos, alargando a su vez el entrenamiento.

- `D_FF`: Siguiendo la arquitectura estándar propuesta por Vaswani et al. (2017), la dimensión de la red *feed-forward* se establece en $d_{ff} = 4d = 2048$.

- `N_HEADS`: De nuevo, se ha empleado la configuración del artículo original. Emplear 8 cabezales significa que el modelo proyecta los ítems en subespacios de dimensión $d_k = 64$, permitiendo que cada cabezal se especialice en capturar distintos tipos de relaciones (por ejemplo, uno podría enfocarse en las similitudes por género cinematográfico, y otro enfocarse en las distancias temporales).

- `NUM_LAYERS`: Dada la naturaleza del conjunto de datos y la longitud de las secuencias, se determinó que $L = 2$ capas de *encoder* son suficientes para lograr la abstracción necesaria sin requerir un coste computacional excesivo. Se experimentó con 4, 6 y 12 capas, sin que ofrecieran una mejora significativa en métricas.

- `DROPOUT`: Se aplica un *dropout* del 10 % en las capas de atención y FF para mejorar la robustez del modelo. Este parámetro (fundamentado en la arquitectura original, véase Vaswani et al., 2017) controla cuántas neuronas apaga aleatoriamente la capa Dropout. De esta manera, obliga a la red a no depender de conexiones específicas, mejorando su capacidad de generalización.

3.3. Mamba

A pesar del éxito indiscutible de la arquitectura Transformer en múltiples ámbitos, esta presenta un cuello de botella computacional: debido a su mecanismo de atención, el tiempo de entrenamiento escala de forma cuadrática respecto a la longitud de la secuencia. Para mitigar este problema, especialmente en contextos donde los historiales de usuario pueden ser muy extensos, han surgido varias alternativas como xLSTM o los *state space models* (SSM). En particular, Mamba es un tipo de arquitectura SSM presentada a finales del año 2023 (véase Gu y Dao, 2024).

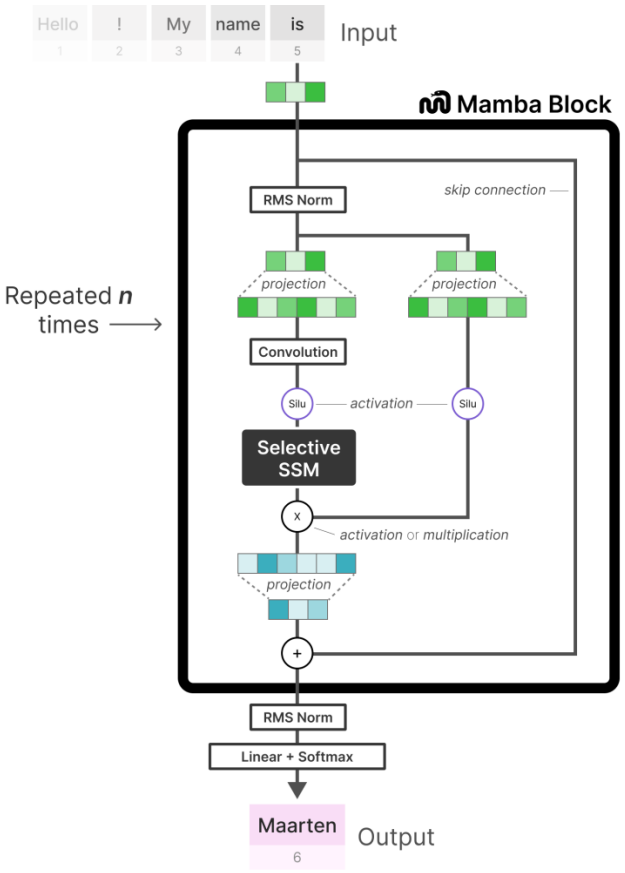

Figura 3.2: Esquema del funcionamiento de un bloque Mamba. Se detalla la expansión dimensional, la convolución local con activación SiLU y el núcleo del SSM selectivo, cuya memoria interna se actualiza dinámicamente. Imagen obtenida de Grootendorst (2024).

El modelo se presenta como una pila de bloques Mamba. Cada bloque procesa el historial de películas de izquierda a derecha: al igual que con las clásicas RNNs y LSTMs, este modelo posee una noción natural del orden cronológico. Cuenta con una memoria interna (llamada *estado oculto*) que le permite comprimir la información "antigua" (por ejemplo, la información relativa a los primeros ítems para predecir el último). La gran innovación del modelo con respecto a esta memoria interna fue hacerla selectiva: a diferencia de modelos como las LSTMs que actualizaban su memoria siempre de la misma manera, Mamba puede "razonar" sobre la marcha, a través de un mecanismo de selección de información.

Gracias a formularse de manera similar a una RNN, Mamba logra estabilizar el tiempo de entrenamiento a medida que el parámetro N (número máximo de ítems en la secuencia) aumenta. En contraposición con el Transformer, que escala de forma cuadrática, Mamba lo hace de forma lineal. Esta es una gran ventaja en contextos donde es fundamental tener historiales largos: de hecho, es uno de las grandes problemáticas a las que se enfrentan tecnológicas como OpenAI o Google en el momento de entrenar sus modelos de IA.

3.3.1. Modelado

De la misma manera que en el Transformer, el objetivo es transformar el historial cronológico de interacciones de un usuario, $S^u = (s_1, s_2, \ldots, s_n)$, en una representación que el modelo pueda emplear

para la predicción del siguiente ítem. Recordando lo presentado en la Sección 3.2, el Transformer necesitaba dos matrices de *embeddings*: una semántica $\mathbf{M}$ y otra de posición $\mathbf{P}$. Esta última ya no será necesaria con Mamba, ya que su formulación procesa intrínsecamente la información de manera secuencial.

Así pues, el primer paso consiste en mapear cada identificador de ítem $i \in \mathcal{I}$ a un espacio vectorial de dimensión d. Para ello, se define una matriz de *embeddings* $\mathbf{E} \in \mathbb{R}^{|\mathcal{I}| \times d}$, la cual se actualiza durante la fase de entrenamiento mediante retropropagación. La representación inicial de un ítem en la posición t de la secuencia, denotada como $\boldsymbol{x}_t^{(0)} \in \mathbb{R}^d$, viene dada de forma directa por su *embedding* (inicializado de forma aleatoria):

$$\boldsymbol{x}_t^{(0)} = \mathbf{E}_{s_t}$$

Agrupando los vectores individuales se construye la matriz inicial $\mathbf{X}^{(0)} \in \mathbb{R}^{N \times d}$, siendo N la longitud máxima de la secuencia. Esta matriz constituye la entrada para la pila de bloques Mamba.

Bloque mamba

Dentro de cada bloque, la matriz $\mathbf{X}^{(l-1)}$ atraviesa primero una capa de normalización (véase la Ecuación 1.7), estandarizando las activaciones.

$$\hat{\mathbf{X}}^{(l-1)} = \text{LayerNorm}(\mathbf{X}^{(l-1)}).$$

Seguidamente, se divide el flujo de información en dos ramas: una conexión residual $\mathbf{Z}$ y la entrada principal $\mathbf{X}'$. Esto se efectúa proyectando la entrada a un espacio de mayor dimensión mediante un factor de expansión (`EXPAND`, normalmente toma el valor 2):

$$\mathbf{Z} = \hat{\mathbf{X}}^{(l-1)}\mathbf{W}_z,$$
$$\mathbf{X}' = \hat{\mathbf{X}}^{(l-1)}\mathbf{W}_x,$$

donde $\mathbf{W}_z, \mathbf{W}_x \in \mathbb{R}^{d \times d_{exp}}$ son matrices de pesos aprendibles, y $d_{exp} = d \cdot \texttt{EXPAND}$. El propósito de incrementar las dimensiones es mejorar la capacidad de representación del modelo, de manera análoga a los bloques *feed-forward* (véase Ecuación 3.3). A continuación, se aplica sobre la rama principal $\mathbf{X}'$ una convolución con tamaño de ventana definido por d_{conv} (por ejemplo, $d_{conv} = 4$), seguida de una función de activación no lineal (SiLU, véase la Ecuación 1.3). Esto permite a cada ítem extraer información de sus ítems adyacentes en el historial. Para una secuencia $\boldsymbol{x}$ y unos pesos aprendibles $\boldsymbol{w} \in \mathbb{R}^{d_{conv}}$, la convolución en el instante de tiempo t se formula como:

$$\text{Conv}(\boldsymbol{x})_t = \sum_{k=0}^{d_{conv}-1} w_k x_{t-k}. \tag{3.9}$$

Esta operación se aplica de forma independiente a cada una de las d_{exp} dimensiones, obteniendo así la matriz $\tilde{\mathbf{X}}$:

$$\tilde{\mathbf{X}} = \text{SiLU}(\text{Conv}(\mathbf{X}')).$$

Paralelamente, se aplica la misma función de activación sobre la conexión residual:

$$\tilde{\mathbf{Z}} = \text{SiLU}(\mathbf{Z}).$$

SSM selectivo

Tras esta fase, las información pasa al núcleo del bloque Mamba: el SSM selectivo. Un SSM clásico utiliza una entrada $x(t) \in \mathbb{R}$ y un estado oculto $\boldsymbol{h}(t) \in \mathbb{R}^n$ (con $\boldsymbol{h}(0) = \mathbf{0}$) para obtener la representación $y(t)$:

$$\boldsymbol{h}'(t) = \mathbf{A}\boldsymbol{h}(t) + \mathbf{B}x(t),$$
$$y(t) = \mathbf{C}\boldsymbol{h}(t),$$

donde $\mathbf{A} \in \mathbb{R}^{n \times n}$ es la matriz de evolución del estado, y $\mathbf{B} \in \mathbb{R}^{n \times 1}$, $\mathbf{C} \in \mathbb{R}^{1 \times n}$ son matrices de proyección. Este estado oculto $\boldsymbol{h}(t)$ es el que actúa como la "memoria" del modelo. En el caso de Mamba, estas dos operaciones se aplican de forma independiente para cada una de las d_{exp} dimensiones de $\tilde{\mathbf{X}}$.

Sin embargo, este SSM no se puede utilizar así como así. Para que el modelo pueda procesar las secuencias (que son de naturaleza discreta), se requiere un paso previo. Esto se puede hacer de varias formas, aunque en el artículo original, Gu y Dao (2024) proponen usar el *zero-order-hold* (ZOH):

$$\bar{\mathbf{A}} = \exp(\Delta\mathbf{A}),$$
$$\bar{\mathbf{B}} = (\Delta\mathbf{A})^{-1}(\exp(\Delta\mathbf{A}) - \mathbf{I}) \cdot \Delta\mathbf{B},$$

donde Δ es el tamaño de paso. De esta manera, podemos expresar el SSM Selectivo como una formulación recurrente que actualiza el estado paso a paso, de manera análoga a las RNNs:

$$\boldsymbol{h}_t = \bar{\mathbf{A}}\boldsymbol{h}_{t-1} + \bar{\mathbf{B}}\tilde{x}_t,$$
$$y_t = \mathbf{C}\boldsymbol{h}_t,$$

donde $\tilde{x}_t$ representa el valor del ítem en el instante t en una dimensión determinada de d_{exp}.

La innovación de Mamba con respecto a un SSM tradicional consistió en hacer este mecanismo selectivo. A diferencia de un SSM clásico, donde $\mathbf{B}$ y $\mathbf{C}$ son invariantes en el tiempo, Mamba parametriza estas matrices (junto con el tamaño de paso Δ) para que dependan de $\tilde{x}_t$. Así, $\mathbf{B}_t$, $\mathbf{C}_t$ y Δ_t se obtienen mediante proyecciones lineales de los datos de entrada en cada instante t. En el contexto de recomendación, esto convierte al modelo en uno capaz de razonar qué información memorizar y cuál olvidar. Por ejemplo, si un usuario visualiza un ítem muy importante, un Δ_t elevado podría obligar a una actualización drástica de la memoria $\boldsymbol{h}_t$ (y viceversa).

Finalmente, las salidas y_t del SSM selectivo se agrupan en una matriz $\mathbf{Y} \in \mathbb{R}^{N \times d_{exp}}$ y se pasan por una capa de *dropout*. Seguidamente, esta matriz se multiplica elemento a elemento con la conexión residual $\tilde{\mathbf{Z}}$ y se proyecta de vuelta a la dimensión original d:

$$\mathbf{X}^{(l)} = (\text{Dropout}(\mathbf{Y}) \odot \tilde{\mathbf{Z}})\mathbf{W}_{mix}, \tag{3.10}$$

siendo $\mathbf{W}_{mix} \in \mathbb{R}^{d_{exp} \times d}$ una matriz de pesos. Este proceso se repite hasta obtener la salida del último bloque Mamba, $\mathbf{X}^{(L)}$.

Capa de salida y predicción

Como sugieren Gu y Dao (2024) en el artículo original, se aplica una capa LayerNorm (véase la Ecuación 1.7) a la salida del último bloque, $\mathbf{X}^{(L)}$, para obtener las representaciones finales de cada ítem en la secuencia de usuario:

$$\mathbf{X}^{(L)}_{\text{final}} = \text{LayerNorm}(\mathbf{X}^{(L)}).$$

Al igual que en el modelo Transformer, se toma el vector correspondiente a la última posición de la secuencia (embebido con la información de toda la cronología), $\boldsymbol{x}_N^{(L)} \in \mathbb{R}^d$, y se proyecta mediante una capa lineal (sin sesgo b) al tamaño del vocabulario $|\mathcal{I}|$:

$$\hat{\boldsymbol{y}} = \boldsymbol{x}_N^{(L)}\mathbf{W}_{out}, \tag{3.11}$$

con $\mathbf{W}_{out} \in \mathbb{R}^{d \times |I|}$ la matriz de pesos aprendibles. El vector resultante $\hat{\boldsymbol{y}} \in \mathbb{R}^{|\mathcal{I}|}$ contiene las puntuaciones brutas para cada ítem. Se aplica una última capa SoftMax (véase Ecuación 1.4) para transformar estas puntuaciones en probabilidades. De esta manera se obtiene una lista del catálogo de películas en la que cada una tiene una probabilidad asignada.

3.3.2. Entrenamiento

Este modelo se entrena de manera similar al Transformer (véase la Subsección 3.2.2), minimizando también la función de pérdida $\mathcal{L}_{CE}$. Sin embargo, durante esta fase surge una diferencia fundamental en la implementación computacional. Mientras que el Transformer requiere el uso de una máscara causal (la matriz **MASK**) para evitar que la secuencia $(s_1, s_2, \ldots, s_r)$ pueda ver los ítems $(s_{r+1}, \ldots, s_t)$, Mamba no la precisa. Como los modelos basados en SSMs ya procesan la información de manera causal (de izquierda a derecha), la arquitectura garantiza por diseño que la predicción en el instante $r+1$ depende única y exclusivamente del historial $\mathbf{S}^u_{1:r}$.

¿Y por qué no usar RNNs clásicas, ya que son recurrentes y no precisan una máscara causal? Además de otros problemas como el desvanecimiento del gradiente, el gran problema de la recurrencia es que, a priori, no es una operación paralelizable. Aunque no entraremos en detalles, gran parte del éxito de Mamba fue lograr paralelizar esta recurrencia durante el entrenamiento, a través del llamado *selective scan* (véase Gu y Dao, 2024).

3.3.3. Configuración de hiperparámetros

La selección de hiperparámetros se hizo teniendo un cuenta dos aspectos: la continuidad con el modelo ajustado anteriormente (el Transformer) y la optimización de características propias a Mamba. Tras una fase de experimentación, se establecieron los valores detallados en el Tabla 3.3.

Hiperparámetro	Valor	Descripción
`D_MODEL` (d)	512	Dimensión del espacio de *embedding*.
`NUM_LAYERS` (L)	2	Número de bloques Mamba apilados.
`D_CONV` (d_{conv})	8	Tamaño de la ventana de la convolución local.
`EXPAND`	2	Factor de expansión de dimensiones.
`DROPOUT`	0.1	Tasa de regularización por abandono.

Tabla 3.3: Configuración de hiperparámetros para el modelo Mamba.

La elección de estos valores se fundamenta en los siguientes criterios:

- `D_MODEL`: Con el objetivo de comparar métricas de una forma más justa, se estableció $d = 512$ por ser consistentes con el modelo Transformer. A pesar de ello, se observó que Mamba es mucho más robusto que el Transformer para tamaños pequeños de *embeddings* (véase la Tabla 3.4), por lo que podría ser más adecuado en casos donde la GPU se viera limitada.
- `NUM_LAYERS`: al igual que con el modelo Transformer, se estableció en 2 capas, puesto que un número de capas mayor no ofreció una mejora significativa.
- `D_CONV`: Determina la amplitud de la convolución previa al SSM selectivo. Al establecer $d_{conv} = 8$, el modelo considera ventanas locales de 8 ítems consecutivos, capturando así relaciones a corto plazo. Aunque en la arquitectura original propuesta por Gu y Dao (2024) se establece $d_{conv} = 4$ como estándar, en este trabajo se optó por ensanchar la ventana a 8 ítems. Esto se justifica por la naturaleza de la tarea de recomendación, ya que aumentar la ventana permite capturar mejor información a corto-medio plazo.
- `EXPAND`: Este factor es el que ensancha temporalmente la dimensión de los datos dentro del bloque Mamba antes de aplicar el SSM selectivo. Resulta algo análogo a la operación que

realiza la expansión que realiza la red *feed-forward* en un Transformer (véase la Ecuación 3.3), proporcionando una mayor riquez expresiva. Se estableció en `EXPAND`=2 según la recomendación del artículo original.

Dim (d)	**HR@1**	**HR@20**	**HR@300**	**MRR@300**	**Tiempo (min)**
16	3.99	24.2	70.0	0.0839	5.15
32	5.82	25.0	70.8	0.1036	3.62
64	6.54	27.8	72.0	0.1140	2.87
128	7.18	28.6	73.2	0.1234	2.35
256	7.42	31.3	71.9	0.1313	2.57
512	7.34	31.4	72.0	0.1279	3.55
1024	6.78	29.9	71.1	0.1180	6.62

Tabla 3.4: Rendimiento del modelo Mamba en función de la dimensión de los *embeddings* d. Valores mayores para las columnas 2 a 5 se asocian con mejores rendimientos. La última columna indica el tiempo precisado para entrenar la red neuronal.

3.4. xLSTM

En el capítulo anterior, hemos visto que Mamba se presentó como una alternativa a los modelos Transformer, paliando uno de sus grandes cuellos de botella computacionales: el tiempo de entrenamiento escala de forma cuadrática con respecto a la longitud de la secuencia. Unos meses más tarde de la publicación de Mamba, se dio el resurgimiento de una de las redes neuronales más exitosas antes de los Transformer: las *long short-term memory* (LSTM, véase Hochreiter y Schmidhuber, 1997). Presentada originalmente a finales de los noventa, la arquitectura LSTM dominó el *sequence modeling* durante años, pero fue perdiendo protagonismo frente a los Transformer debido a dos grandes limitaciones:

- La memoria de una LSTM clásica es escalar: comprime toda la información histórica en un vector de tamaño fijo. Esto le impide almacenar y recuperar detalles complejos cuando la secuencia es demasiado larga.
- Debido a su naturaleza recurrente, el cálculo del cada estado en la secuencia h_t depende del anterior, h_{t-1}. Esto impide la paralelización durante el entrenamiento.

Para solucionar estos problemas, surgió la arquitectura xLSTM (*extended long short-term memory*, véase Beck et al., 2024). Esta arquitectura introdujo dos grandes innovaciones: el *gating* exponencial y nuevas estructuras de memoria. En concreto, el artículo original propone dos clases de bloques dentro del modelo: sLSTM (*scalar LSTM*) y mLSTM (*matrix LSTM*). Nos centraremos en estos últimos, diseñados específicamente para superar las dificultades computacionales de las LSTM.

3.4.1. Modelado

Al igual que en las arquitecturas anteriores, partimos del historial cronológico de interacciones del usuario $S^u = (s_1, s_2, \ldots, s_n)$, donde cada s_t representa una película en $\mathcal{I}$. El objetivo será transformar esta secuencia en una representación vectorial que permita predecir el siguiente ítem a consumir.

La naturaleza recurrente del xLSTM permite prescindir de *embeddings* de posición. Así pues, se mapea cada identificador de película $i \in \mathcal{I}$ a un espacio vectorial de dimensión d a través de la matriz

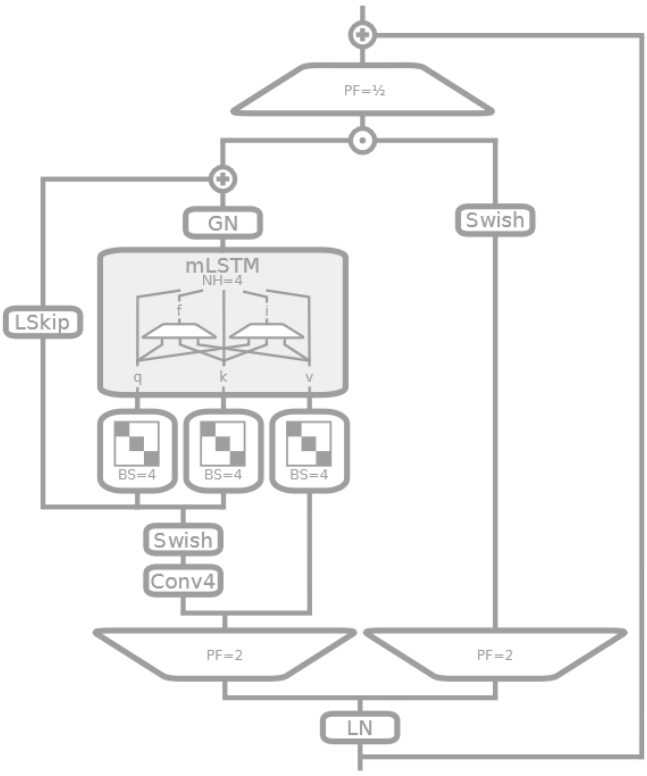

Figura 3.3: Esquema de un bloque mLSTM dentro de la arquitectura xLSTM. Se muestra el flujo con las proyecciones de consulta ($\boldsymbol{q}_t$), clave ($\boldsymbol{k}_t$) y valor ($\boldsymbol{v}_t$), además de las puertas de entrada y olvido. Imagen extraída de Beck et al. (2024).

de *embeddings* $\mathbf{E} \in \mathbb{R}^{|\mathcal{I}| \times d}$, aprendible durante el entrenamiento. La representación inicial de un ítem s_t en la secuencia, $\boldsymbol{x}_t^{(0)} \in \mathbb{R}^d$, viene dada por el *embedding* correpondiente a la película que representa:

$$\boldsymbol{x}_t^{(0)} = \mathbf{E}_{s_t}.$$

Bloque mLSTM

El núcleo del modelo es el bloque mLSTM. Este comienza aplicando una capa de normalización a la matriz de vectores iniciales $\mathbf{X}^{(0)}$ (o proveniente del bloque anterior $\mathbf{X}^{(l-1)}$):

$$\hat{\mathbf{X}}^{(l-1)} = \text{LayerNorm}(\mathbf{X}^{(l-1)}),$$

para después ser dividido en dos ramas: una conexión residual $\mathbf{Z}$ y la entrada principal $\mathbf{X}'$. Estas son proyectadas a un espacio de mayor dimensión mediante un factor de expansión (`EXPAND`, normalmente toma el valor 2):

$$\mathbf{Z} = \hat{\mathbf{X}}^{(l-1)}\mathbf{W}_z$$
$$\mathbf{X}' = \hat{\mathbf{X}}^{(l-1)}\mathbf{W}_x$$

donde $\mathbf{W}_z, \mathbf{W}_x \in \mathbb{R}^{d \times d_{exp}}$ son matrices de pesos aprendibles, y $d_{exp} = d \cdot \texttt{EXPAND}$. Para cada ítem s_t en la secuencia, se emplea un mecanismo de atención similar al del Transformer, calculando las consultas

($\boldsymbol{q}_t$), claves ($\boldsymbol{k}_t$) y valores ($\boldsymbol{v}_t$):

$$\begin{aligned}\boldsymbol{q}_t &= \mathbf{W}_q \boldsymbol{x}_t \in \mathbb{R}^d,\\ \boldsymbol{k}_t &= \mathbf{W}_k \boldsymbol{x}_t \in \mathbb{R}^d,\\ \boldsymbol{v}_t &= \mathbf{W}_v \boldsymbol{x}_t \in \mathbb{R}^d,\end{aligned}$$

donde $\boldsymbol{x}_t$ denota el vector correspondiente a s_t en $\mathbf{X}'$, y $\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v \in \mathbb{R}^{d\times d}$ son matrices de pesos entrenables. A su vez, xLSTM introduce tres "puertas" cuya tarea es controlar el flujo de información.

- **Puerta de entrada (i_t):** Determina qué información será almacenada en la memoria matricial.
- **Puerta de olvido (f_t):** Controla el decaimiento de la memoria acumulada hasta el instante anterior.
- **Puerta de salida (o_t):** Filtra la información que será enviada a la siguiente capa del modelo.

Estas puertas se definen como proyecciones lineales a través de pesos entrenables, seguidas de funciones de activación sigmoides o exponencial:

$$\begin{aligned}i_t &= \exp(\mathbf{W}_i \boldsymbol{x}_t + \boldsymbol{b}_i),\\ f_t &= \text{sigmoide}(\mathbf{W}_f \boldsymbol{x}_t + \boldsymbol{b}_f),\\ o_t &= \text{sigmoide}(\mathbf{W}_o \boldsymbol{x}_t + \boldsymbol{b}_o).\end{aligned}$$

El uso de la función de activación exponencial para la puerta de entrada es lo que permite gestionar de forma más eficaz las dependencias a largo plazo en comparación con las LSTM clásicas.

Siguiendo la filosofía del Transformer, el bloque mLSTM emplea una estructura multicabezal. Tras el cálculo de las proyecciones $\boldsymbol{q}_t, \boldsymbol{k}_t, \boldsymbol{v}_t$ y las puertas i_t, f_t, estas se dividen en h cabezales independientes de dimensión $d_k = d/h$ (por ejemplo, $h = \texttt{N\_HEADS} = 8$). Una vez procesada la información en cada cabezal, los resultados se concatenan para recuperar la dimensión original d antes de la fase final de *gating* de salida.

Por otro lado, el bloque mLSTM sustituye la memoria escalar (tradicional en las LSTM, véase Hochreiter y Schmidhuber, 1997) por una memoria matricial. De esta manera se incrementa enormemente la capacidad para almacenar y relacionar información a lo largo de la secuencia. Esto se hace a través de una matriz $\mathbf{C}_t \in \mathbb{R}^{d\times d}$, cuya actualización en cada paso de tiempo t se define mediante la siguiente recurrencia:

$$\mathbf{C}_t = f_t \mathbf{C}_{t-1} + i_t(\boldsymbol{v}_t \boldsymbol{k}_t^\top), \tag{3.12}$$

donde el término $f_t\mathbf{C}_{t-1}$ representa el olvido del historial pasado y el producto exterior $i_t(\boldsymbol{v}_t\boldsymbol{k}_t^\top)$ representa la nueva información asociada al ítem. Finalmente, la salida del bloque se realiza proyectando la consulta $\boldsymbol{q}_t$ sobre la memoria acumulada:

$$\tilde{\boldsymbol{h}}_t = \mathbf{C}_t \boldsymbol{q}_t.$$

Sin embargo, entrenar una red neuronal recurrente puede ser excesivamente lento si carecemos de alguna forma de paralelizar. Es posible reformular las operaciones anteriores como una operación global: considerando la secuencia completa, la salida en el instante t puede expresarse como:

$$\tilde{\boldsymbol{h}}_t = \sum_{j=1}^{t} \left(\prod_{k=j+1}^{t} f_k \right) i_j (\boldsymbol{k}_j^\top \boldsymbol{q}_t) \boldsymbol{v}_j. \tag{3.13}$$

Esta operación es en cierta manera equivalente a un mecanismo de *atención lineal* (véase Katharopoulos et al., 2020), donde el peso de cada ítem pasado viene determinado por un decaimiento acumulado (el

producto de las puertas de olvido f_k).

Finalmente, se aplica la puerta de salida o_t a la memoria matricial $\tilde{\boldsymbol{h}}_t$ para filtrar la información relevante. La salida final del bloque mLSTM, $\boldsymbol{h}_t$, recupera la conexión residual Z y una capa de *dropout* (véase la Ecuación 1.6):

$$\boldsymbol{h}_t = \text{Dropout}(\mathbf{W}_{doors}(\tilde{\boldsymbol{h}}_t \odot o_t)) + \boldsymbol{z}_t$$

donde $\odot$ representa el producto elemento a elemento, $\mathbf{W}_{doors}$ es una matriz de pesos entrenable y $\boldsymbol{z}_t$ es el vector correspondiente a la posición t de la conexión residual $\mathbf{Z}$. Este proceso se repite para cada uno de los L bloques mLSTM, obteniendo una representación final $\boldsymbol{h}_t^{(L)}$. Agrupando las representaciones para cada ítem en la secuencia se obtiene la matriz $\mathbf{H}^{(L)}$.

Capa de salida y predicción

Siguiendo el ejemplo de Mamba (véase la Sección 3.3), se aplica una normalización final sobre la salida del último bloque antes de realizar la inferencia:

$$\mathbf{H}_{\text{final}}^{(L)} = \text{LayerNorm}(\mathbf{H}^{(L)}).$$

Al igual que en las arquitecturas presentadas anteriormente, para la tarea de recomendación se selecciona el vector correspondiente a la última posición de la secuencia, $\tilde{\boldsymbol{h}}_N^{(L)} \in \mathbb{R}^d$, y se proyecta mediante una capa lineal al espacio del vocabulario de ítems $|\mathcal{I}|$:

$$\hat{\boldsymbol{y}} = \boldsymbol{h}_N^{(L)} \mathbf{W}_{out},$$

donde $\mathbf{W}_{out} \in \mathbb{R}^{d \times |\mathcal{I}|}$ es una matriz de pesos aprendible. El vector resultante $\hat{\boldsymbol{y}}$ contiene las puntuaciones para cada ítem del catálogo. Por último, se aplica una capa SoftMax (véase la Ecuación 1.4), para transformar estas puntuaciones en probabilidades. El modelo estima así la probabilidad de que un usuario consuma cada ítem, condicionada a su historial (véase Goodfellow et al., 2016):

$$\hat{y}_i = \hat{p}_i \approx \mathbb{P}(s_{t+1} = i \mid S_{1:t}^u).$$

3.4.2. Entrenamiento

El proceso de entrenamiento de la arquitectura xLSTM comparte su fundamento teórico con los modelos presentados en las secciones anteriores (véanse la Sección 3.2 y la Sección 3.3). Así, se repite toda la lógica ya descrita, minimizando la función de pérdida de entropía cruzada ($\mathcal{L}_{CE}$).

Sin embargo, llevar la arquitectura xLSTM a la práctica supuso un mayor desafío. Al tratarse de un modelo tan reciente (el artículo original de Beck et al., 2024 fue publicado en el año 2024), el ecosistema computacional de librerías y dependencias no está preparado para cualquier unidad gráfica. En particular, la GPU usada (NVIDIA Tesla T4) no cumple los requisitos necesarios para ejecutar los bloques mLSTM. De esta manera, se ha optado por una implementación "manual" que no aprovecha los algoritmos de paralelización implementados originalmente. Esto es importante, pues la gran ventaja que presentan los xLSTM sobre los Transformer es, al igual que Mamba, la escalabilidad del tiempo de entrenamiento con respecto a la longitud de secuencia N. Como veremos en el Capítulo 5, esta implementación manual repercute en las mediciones de tiempos de entrenamiento sin afectar a las métricas, pues se mantienen las operaciones de la implementación teórica.

3.4.3. Configuración de hiperparámetros

Se han configurado los hiperparámetros para mantener una cierta paridad con los modelos Transformer y Mamba, y para optimizar el funcionamiento del propio xLSTM. Los valores seleccionados se detallan en la Tabla 3.5.

Hiperparámetro	Valor	Descripción
`D_MODEL` (d)	512	Dimensión del espacio de *embedding*.
`NUM_LAYERS` (L)	2	Número de bloques mLSTM apilados.
`N_HEADS` (h)	8	Número de cabezas de la memoria matricial.
`EXPAND`	2	Factor de expansión de dimensiones internas.
`DROPOUT`	0.1	Tasa de regularización por abandono.

Tabla 3.5: Configuración de hiperparámetros para el modelo xLSTM.

La elección de estos valores se fundamenta en los siguientes criterios:

- `D_MODEL`: Se mantiene en 512 para asegurar que la capacidad de representación sea equivalente a la del Transformer y Mamba.
- `N_HEADS`: De acuerdo con lo empleado para el Transformer, y siguiendo la recomendación original de Vaswani et al. (2017), se establecieron $h = 8$ cabezales. Esto significa que el modelo proyecta cada ítem en subespacios de dimensión $d/h = 64$.
- `EXPAND`: Se establece un factor de expansión de 2, siguiendo lo establecido por Beck et al. (2024). Esta proyección a $d_{exp} = 1024$ dimensiones antes de la actualización de la memoria matricial permite que el modelo pueda capturar interacciones no lineales más complejas sin comprometer excesivamente el tiempo de entrenamiento.
- `NUM_LAYERS`: Se utilizan 2 bloques mLSTM apilados. De la misma forma que en los experimentos anteriores, aumentar la profundidad más allá de 2 capas no implicó mejoras significativas en cuestión de métricas, pero sí incrementó el tiempo de entrenamiento.
- `DROPOUT`: Al igual que para el Transformer y Mamba, se aplica una tasa del 10 % para asegurar que el aprendizaje sea generalizable a ítems no vistos durante el entrenamiento.

3.5. BERT y modelos híbridos

Hasta este punto, se han presentado arquitecturas de modelado secuencial puramente colaborativas (véase la Sección 1.1). Modelos como el Transformer, Mamba o xLSTM procesan historiales de usuario donde cada película está representada únicamente por un *embedding* de ítem, y otro de posición en el caso del Transformer. Estos modelos logran captar patrones secuenciales basándose en las interacciones cronológicas, pero carecen de información intrínseca sobre el contenido de los ítems. En una arquitectura estándar, dos películas con temáticas idénticas son tratadas como entidades completamente independientes a menos que los usuarios las consuman de manera conjunta reiteradamente.

Entonces, ¿cómo se puede dotar al modelo secuencial de información adicional? Este problema ha sido atacado de diversas maneras a lo largo de los últimos años. Uno de los artículos más influyentes en el sector industrial, *Deep neural networks for Youtube recommendations* (véase Covington et al., 2016), evidenció la necesidad de alimentar a las redes neuronales profundas con información mas allá de un simple historial. Como se observa en la Figura 3.4, esto se puede realizar concatenando metadatos de usuario como la posición geográfica o la edad a los *embeddings* del historial cronológico.

Sin embargo, el *dataset* de MovieLens carece de información intrínseca de usuario. Por ello, precisamos hacer algo similar con la información de los ítems, tratando cada variable de manera individual: la popularidad y puntuación media como variables numéricas normalizadas, los géneros como variables discretas tomando valores en $\{0, 1\}$ (*one-hot-encoding*), etcétera. Con la reciente explosión del

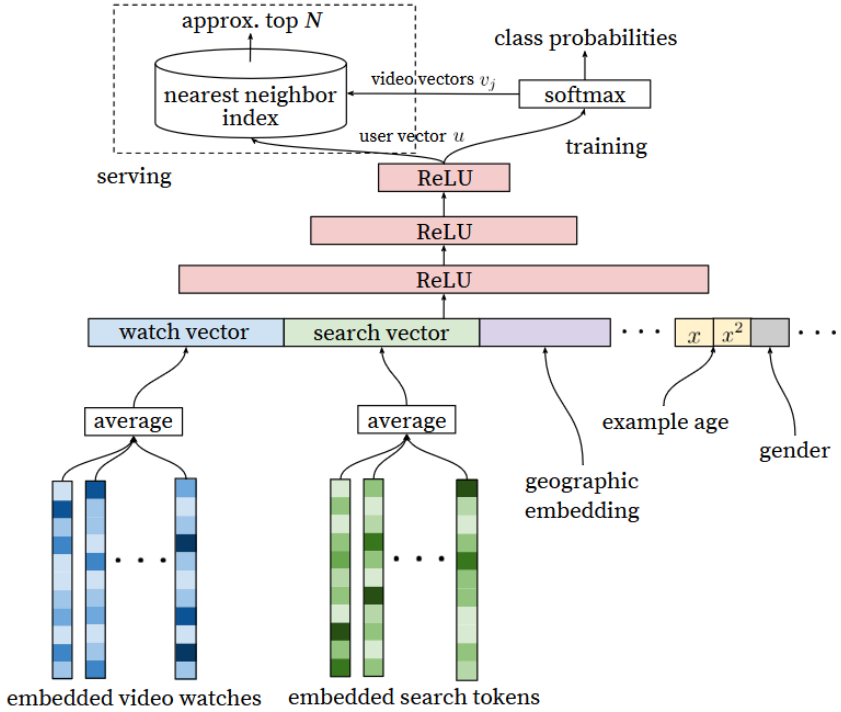

Figura 3.4: Esquema del funcionamiento de la red neuronal para la recomendación de vídeos en Youtube en el año 2016. Los *embeddings* correspondientes al historial cronológico de usuario son concatenados con información adicional: búsquedas, posición geográfica, edad y género. Imagen extraída de Covington et al. (2016).

procesamiento de lenguaje natural, se nos presenta una alternativa mucho más potente: permitir que un modelo lea, entienda y codifique automáticamente toda esta información. Así, se ha recurrido a la familia de modelos BERT (*bidirectional encoder representations from transformers*, véase Devlin et al., 2019). Estos son una clase de modelos Transformer en los que se utiliza un *encoder* bidireccional (véase la Sección 3.2). Presentan la particularidad de ser fácilmente refinados, ya sea para tareas de respuesta a preguntas o inferencia de lenguaje, sin necesidad de entrar en modificaciones sustanciales de su arquitectura (tan solo añadiendo una nueva capa final).

3.5.1. TF-IDF

BERT no es el primer modelo basado en contenido. Al igual que Item-KNN es un algoritmo clásico de filtrado colaborativo, existen otros métodos tradicionales que toman información del contexto de cada ítem. Aquí entra el modelo TF-IDF (*Term Frequency - Inverse Document Frequency*, véase Salton y McGill, 1983). Este evalúa la importancia de una palabra dentro de un documento concreto en relación con un corpus global de textos. En este caso, el "documento" se correspondería con la información disponible sobre una película, y el "corpus" sería el catálogo de películas disponible. A partir de ello se construye una puntuación:

$$\text{TF-IDF}(w, i, \mathcal{I}) = \text{TF}(w, i) \cdot \text{IDF}(w, \mathcal{I}),$$

donde w denota una palabra, i una película e $\mathcal{I}$ el vocabulario. Aunque no entraremos en detalles de su cálculo, el término $\text{TF}(w, i)$ mide la frecuencia con la que la palabra w aparece en el documento de la película i. Por otro lado, $\text{IDF}(w, \mathcal{I})$ evalúa la especificidad de la palabra a lo largo de todo el catálogo. Es decir, $\text{TF-IDF}(w, i, \mathcal{I})$ calcula la importancia de una palabra en una película con relación

al contexto del catálogo: la palabra "ghost" será mucho más importante en el contexto de la película *The Conjuring* que la palabra "The".

Mediante este procedimiento, cada película $i \in \mathcal{I}$ queda representada por un vector formado por las puntuaciones de cada palabra de su contexto, $\boldsymbol{v}_i \in \mathbb{R}^{|\mathcal{W}|}$, donde $|\mathcal{W}|$ es el tamaño del vocabulario total del corpus. Finalmente, para generar una recomendación a un usuario se emplea un procedimiento similar al del Item-KNN (véase la Sección 3.1). Se obtiene el vector promedio entre las películas del usuario (todas o a una ventana fija) y se calcula la similitud coseno (véase la Ecuación 3.1) con los vectores de las películas del vocabulario, seleccionando aquellas con las puntuaciones más elevadas.

3.5.2. Generación del contexto y variantes de BERT

Para que un modelo de lenguaje como BERT pueda extraer información semántica relevante, se debe construir un documento de texto que resuma fielmente el contenido de cada película. Haciendo uso de la API de The Movie Database (2026), se recopilaron los metadatos de todas las películas presentes en el conjunto de datos de MovieLens. Para cada ítem, se generó un párrafo concatenando los siguientes campos: título, año de lanzamiento, géneros, director, actores principales (top 3), clasificación por edades (adulto sí/no), palabras clave y la sinopsis completa.

Este bloque de texto se procesó de manera independiente (fuera del bucle de entrenamiento principal del modelo de recomendación) para obtener un *embedding* estático por cada película. Durante la fase de experimentación, se evaluaron dos variantes de la arquitectura BERT, accesibles a través de la plataforma Hugging Face (véase Hugging Face, 2025):

- **MiniBERT** (Wang et al., 2020): Un modelo ligero compuesto por 22 millones de parámetros y 6 capas ocultas, con una ventana de contexto limitada a 512 *tokens* (aproximadamente 300 palabras). Destaca por su eficiencia computacional y su capacidad para resaltar palabras clave y fraseología similar entre textos descriptivos cortos.
- **ModernBERT** (Warner et al., 2025): Una arquitectura considerablemente más robusta y reciente, equipada con 149 millones de parámetros, 22 capas y una ventana de contexto extendida de hasta 8192 *tokens* (cerca de 6000 palabras). Su principal ventaja radica en la capacidad para capturar matices semánticos mucho más profundos y relaciones complejas dentro del texto proporcionado.

La extracción de estos vectores permite construir una matriz de *embeddings* semánticos pre-entrenados $\mathbf{E}_{\text{text}} \in \mathbb{R}^{|\mathcal{I}| \times d_{\text{text}}}$, donde d_{text} representa la dimensión de salida del modelo de lenguaje. Esta información, previamente congelada para evitar un sobrecoste computacional durante la retro-propagación, se integrará en los modelos secuenciales mediante diversas estrategias de proyección y fusión.

A pesar de que el objetivo principal es usar estos *embeddings* como complemento para arquitecturas más potentes, estos se pueden emplear directamente para ofrecer recomendaciones. Reciclando el concepto de similitud coseno visto en la Sección 3.1, y de forma equivalente al TF-IDF, podemos calcular similitudes entre ítems sin más que efectuar el producto escalar de estos vectores previamente normalizados. En la Figura 3.5 observamos un ejemplo de estas similitudes computadas para ciertas películas.

3.5.3. Modelos híbridos con BERT

Al combinar la naturaleza secuencial de los modelos Transformer, Mamba o xLSTM con la información semántica de las películas, construimos lo que denominaremos *modelos híbridos*. Para ello, se combinarán *embeddings* de ambas partes mediante el siguiente método de fusión.

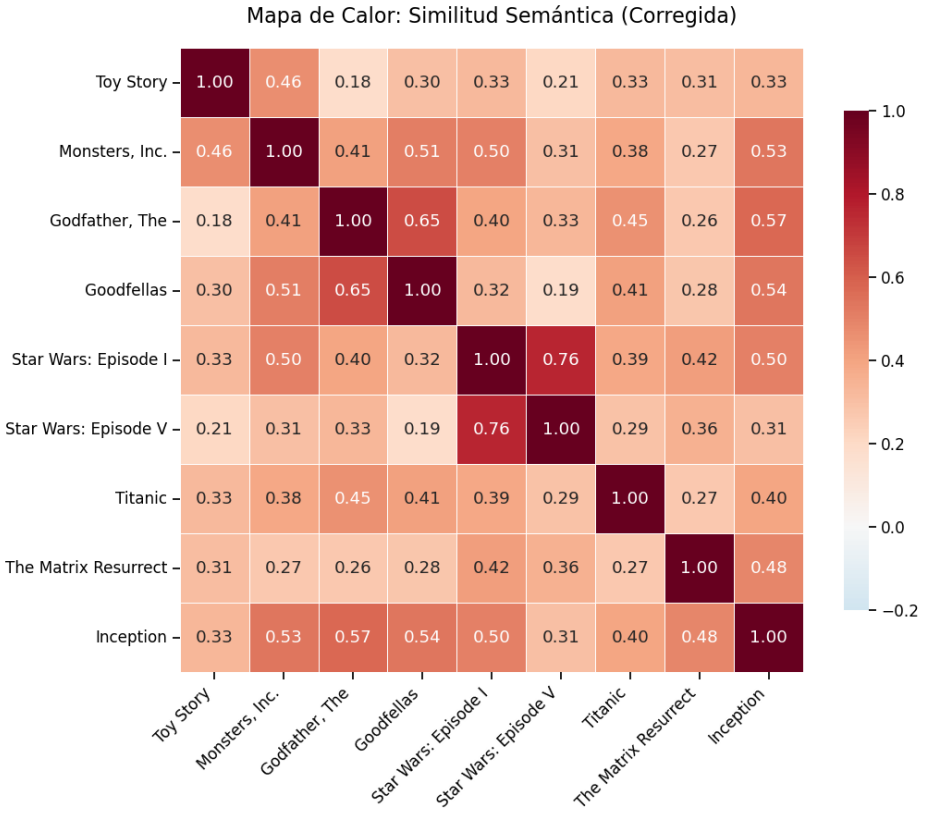

Figura 3.5: Mapa de calor representando las similitudes coseno calculadas para una selección de películas mediante ModernBERT. Valores cercanos a 1 implican un mayor parecido, mientras que valores más bajos denotan una menor relación.

Fusión temprana de ítems

El objetivo es potenciar la representación inicial de cada película en la secuencia antes de que sea procesada por el núcleo del modelo secuencial. Para ello, cada ítem s_t dispone de dos representaciones iniciales distintas:

- Un *embedding* entrenable, $\boldsymbol{e}_{\text{train}} \in \mathbb{R}^d$, que captura las relaciones colaborativas a través de la red neuronal secuencial.
- Un *embedding* de contenido congelado, $\boldsymbol{e}_{\text{BERT}} \in \mathbb{R}^{d_{\text{text}}}$, pre-entrenado mediante ModernBERT o MiniBERT a partir de los metadatos. Un *embedding* se dice congelado si no se actualiza durante el entrenamiento (en la retropropagación).

Para garantizar que ambas representaciones puedan interactuar de forma equilibrada, el vector de BERT se somete a una fase de estabilización y alineación dimensional. En primer lugar, se aplica una capa de normalización, seguida de una proyección lineal hacia el espacio del modelo secuencial:

$$\tilde{\boldsymbol{e}}_{\text{BERT}} = \text{LayerNorm}(\boldsymbol{e}_{\text{BERT}})\mathbf{W}_{up} + \boldsymbol{b}_{up},$$

donde $\mathbf{W}_{up} \in \mathbb{R}^{d_{\text{BERT}} \times d}$ y $\boldsymbol{b}_{up} \in \mathbb{R}^d$ son parámetros aprendibles. A continuación, se combinan ambas representaciones mediante una operación de concatenación:

$$\boldsymbol{c}_t = \text{Concat}(\boldsymbol{e}_{\text{train}}, \tilde{\boldsymbol{e}}_{\text{BERT}}).$$

El vector resultante $\boldsymbol{c}_t \in \mathbb{R}^{2d}$ transita por dos capas de compresión encargadas de reducir la dimensionalidad de manera gradual hasta recuperar la dimensión original d del modelo secuencial.

$$\begin{aligned} \boldsymbol{z}_t &= \text{ReLU}(\mathbf{W}_{f1}\boldsymbol{c}_t + \boldsymbol{b}_{f1}), \\ \boldsymbol{x}_t^{(0)} &= \mathbf{W}_{f2}\boldsymbol{z}_t + \boldsymbol{b}_{f2}, \end{aligned} \tag{3.14}$$

donde $\mathbf{W}_{f1} \in \mathbb{R}^{\frac{4}{3}d \times 2d}$, $\mathbf{W}_{f2} \in \mathbb{R}^{d \times \frac{4}{3}d}$ y sus respectivos sesgos son parámetros aprendibles. Esta transición evita cuellos de botella y pérdidas abruptas de información semántica (véase Covington et al., 2016). La nueva representación híbrida $\boldsymbol{x}_t^{(0)}$ es la que finalmente alimenta la pila de bloques del modelo secuencial (ya sea la atención multicabezal o los bloques Mamba/mLSTM), permitiendo que la red opere sobre un vector que contiene tanto el contexto colaborativo como el semántico de cada ítem.

3.6. Redes neuronales de grafos y modelos híbridos

A diferencia de los modelos secuenciales presentados en las secciones anteriores, que operan sobre el historial cronológico del usuario (es decir, poseen una noción de orden), las redes neuronales de grafos (*graph neural networks* o GNN) trabajan sobre la estructura global de interacciones. Para este contexto donde existen usuarios y películas, los datos serán representados como un grafo bipartito $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, donde los nodos $\mathcal{V} = U \cup I$ representan a los usuarios e ítems, y las aristas $\mathcal{E}$ representan las transacciones registradas.

El objetivo de las GNN es extraer representaciones numéricas (*embeddings*), tanto para películas como ítems, que capten la similitud entre nodos. Estos *embeddings* se pueden usar para ofrecer recomendaciones basadas en usuarios similares (filtrado colaborativo), pero el mayor interés será integrarlos en la construcción de modelos híbridos, donde emplearemos como modelos base los ya vistos modelos secuenciales (véanse las Secciones 3.2, 3.3 y 3.4).

Estas redes operan bajo el principio de *message passing* (paso de mensajes), un proceso iterativo donde cada nodo actualiza su representación comunicándose con su entorno. Conceptualmente, este proceso se divide en dos fases dentro de cada capa de la red:

1. **Fase de agregación (`AGGREGATE`):** El nodo recopila la información (los *embeddings*) de todos sus vecinos inmediatos (aquellos conectados a través de una única arista) y la combina. Dependiendo de la arquitectura, esta agregación puede ser una operación sencilla como una suma o una media, o algo más complejo como una combinación ponderada.
2. **Fase de actualización (`UPDATE`):** El nodo toma el mensaje agregado de sus vecinos y lo combina con su propia representación actual para actualizarla.

En la capa $l+1$, la representación del nodo i, denotada como $\boldsymbol{h}_i^{(l+1)}$, se obtiene mediante la siguiente formulación general:

$$\boldsymbol{h}_i^{(l+1)} = \text{UPDATE}^{(l)}\left(\boldsymbol{h}_i^{(l)}, \text{AGGREGATE}^{(l)}\left(\{\boldsymbol{h}_j^{(l)} : j \in \mathcal{N}(i)\}\right)\right), \tag{3.15}$$

donde $\mathcal{N}(i)$ representa el conjunto de vecinos del nodo $i \in \mathcal{G}$ (ya sea un ítem o un usuario), y las funciones exactas de agregación y actualización definen la arquitectura específica del modelo. La Figura 3.6 proporciona una intuición visual de esta operación.

3.6.1. Modelado

Se han explorado tres de las arquitecturas de redes neuronales de grafos más representativas de la literatura. A continuación se describen formalmente sus mecanismos de agregación y actualización,

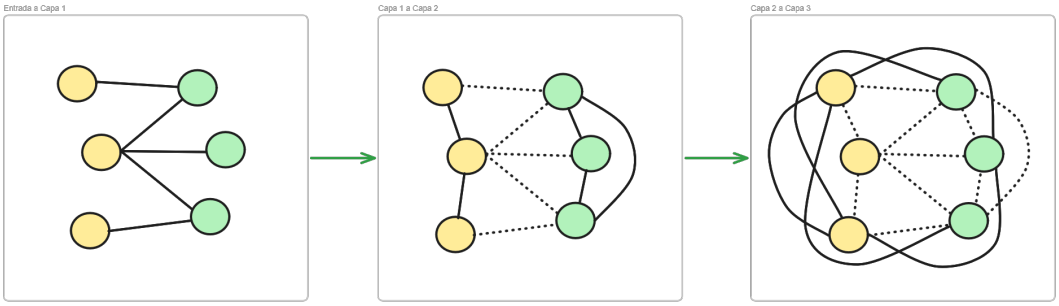

Figura 3.6: Esquema del funcionamiento del *message passing* en una GNN con grafo bipartito (por ejemplo, usuarios y películas). En cada capa, se toman en cuenta los vecinos de cada nodo para construir una nueva representación.

comenzando desde la formulación convolucional clásica (GCN), pasando por una variante optimizada específicamente para tareas de recomendación (LightGCN), y terminando con un enfoque basado en mecanismos de atención (GAT).

GCN

La arquitectura GCN (*graph convolutional network*), propuesta originalmente por Kipf y Welling (2017), adapta el concepto de convolución (empleado en el procesamiento de imágenes) a datos estructurados en grafos. En esta implementación, la operación de *message passing* se define como sigue:

$$\boldsymbol{h}_i^{(l+1)} = \sigma\left(\sum_{j\in\mathcal{N}(i)\cup\{i\}} \frac{1}{\sqrt{|\mathcal{N}(i)\cup\{i\}||\mathcal{N}(j)\cup\{j\}|}}\boldsymbol{h}_j^{(l)}\mathbf{W}^{(l)}\right),$$

donde $\mathbf{W}^{(l)}$ es la matriz de pesos aprendibles de la capa l y σ es una función de activación no lineal. Esta regla de agregación combina la información de forma lineal, ponderando con respecto al grado de cada nodo.

LightGCN

En el ámbito de los sistemas de recomendación, He et al. (2020) demostraron que emplear matrices de pesos entrenables y funciones de activación no lineales podrían ser innecesario a la hora de implementar una GCN. No solo no ofrecen mejoras significativas de rendimiento, sino que pueden empeorarlo. De esta manera nació LightGCN, cuya operación de *message passing* se define como sigue:

$$\boldsymbol{h}_i^{(l+1)} = \sum_{j\in\mathcal{N}(i)} \frac{1}{\sqrt{|\mathcal{N}(i)||\mathcal{N}(j)|}}\boldsymbol{h}_j^{(l)},$$

donde $\boldsymbol{h}_i^{(l)}$ es el *embedding* del nodo i en la capa l. Simplificar una GCN de esta manera permite que el modelo se centre exclusivamente en capturar la proximidad estructural en el grafo. Nótese que, al contrario que en la GCN clásica, no se considera la información relativa al nodo pertinente i en la operación de *message passing*. Esta información no se pierde, sino que vive en el *embedding* inicial, y será recuperada como se detalla a continuación.

La segunda característica distintiva de este modelo es que no utiliza la salida de la última capa de *message passing* como representación final. En su lugar, el *embedding* definitivo de cada usuario y

película se calcula como el promedio de las representaciones obtenidas en todas las capas, incluida la inicial ($l = 0$):

$$\boldsymbol{h}_{final} = \frac{1}{L+1}\sum_{k=0}^{L}\boldsymbol{h}^{(l)},$$

donde L es el número total de capas de convolución. Esto permite tanto mitigar el sobreajuste como capturar relaciones a distintos niveles de cercanía de una manera más directa. Al no contar con matrices de pesos en las convoluciones, los únicos parámetros aprendibles son los *embeddings* de la capa inicial, inicializados de forma aleatoria.

GAT

La arquitectura GAT (*graph attention network*), introducida por Veličković et al. (2018), plantea una mecanismo diferente para la agregación de información en grafos. Mientras que GCN y LightGCN asignan pesos fijos a las conexiones, pues se basan exclusivamente en la estructura (el término $(\sqrt{|\mathcal{N}(i)||\mathcal{N}(j)|})^{-1}$ dependiente del grado de los nodos), GAT emplea un mecanismo de atención para aprender dinámicamente la importancia de cada vecino. Esto significa que el modelo no asume que todas las películas vistas por un usuario tienen el mismo impacto para definir su perfil, lo cual es lógico.

Este mecanismo de atención está inspirado en las novedades traídas por el modelo Transformer de Vaswani et al. (2017). Así pues, usa un sistema de atención multicabezal en cada una de las capas, y unos coeficientes de atención aprendibles α_{ij}, para ponderar la importancia de cada vecino. De esta manera, se define la operación de *message passing* como sigue:

$$\boldsymbol{h}_i^{(l+1)} = \text{Concat}_{k=1}^{H}\left[\sigma\left(\sum_{j\in\mathcal{N}(i)\cup\{i\}}\alpha_{ij}^{k}\mathbf{W}^{k}\boldsymbol{h}_j^{(l)}\right)\right],$$

donde Concat denota la concatenación, H es el número de cabezales, $\mathbf{W}^k$ es la transformación lineal del cabezal k, y σ es la función de activación no lineal. Como se verá en el Capítulo 4, esta arquitectura resulta mucho más potente que las dos anteriores, a expensas de un mayor coste computacional.

3.6.2. Entrenamiento

Los modelos basados en grafos no se entrenan mediante entropía cruzada, sino como un problema de *ranking*. Para ello, se ha empleado la función de pérdida BPR (*bayesian personalized ranking*, véase Rendle et al., 2009). Esta asume que un usuario prefiere un ítem con el que ha interactuado (denominado positivo) frente a uno con el que no (denominado negativo). La función de pérdida a minimizar se define como:

$$\mathcal{L}_{\text{BPR}} = -\frac{1}{N}\sum_{(u,i,j)}\log\sigma(\hat{y}_{ui} - \hat{y}_{uj}), \tag{3.16}$$

donde σ es la función sigmoide, u es el usuario, i es el ítem positivo, j es un ítem negativo muestreado aleatoriamente, e $\hat{y}_{ui} = \boldsymbol{h}_u^\top\boldsymbol{h}_i$ representa la puntuación predicha mediante el producto escalar de sus *embeddings* finales.

3.6.3. Configuración de hiperparámetros

Se ha intentado mantener una concordancia entre las tres implementaciones llevadas a cabo, siendo más difícil comparar la selección de hiperparámetros con el caso de las redes neuronales de procesamiento secuencial. Así pues, se han seleccionado los valores en la Tabla 3.6.

Hiperparámetro	Valor	Descripción
`D_MODEL` (d)	64	Dimensión del espacio de *embedding*.
`NUM_LAYERS` (L)	2	Número de capas de *message passing*.
`N_HEADS` (h)	4	Número de cabezas en la GAT.
`DROPOUT`	0.5	Tasa de regularización por abandono.

Tabla 3.6: Configuración de hiperparámetros para las GNN.

La elección de estos valores se explica a continuación:

- `D_MODEL`: Se establece en 64 por dos razones principales. Por un lado, incrementarlo no supuso ninguna mejora en métricas para los modelos GCN y LightGCN. Por otro, la arquitectura GAT sufre considerables problemas de asignación de memoria, al tener que computar pesos de atención para un grafo con excesivas conexiones (aristas).
- `NUM_LAYERS`: Se utilizan 2 capas de *message passing*. De esta manera, los *embeddings* serán enriquecidos con información de sus vecinos, y de los adyacentes a estos. Limitar la profundidad a dos iteraciones es una manera de evitar el sobreajuste, que llenaría las representaciones de ruido al mezclar información de nodos demasiado lejanos.
- `N_HEADS`: Este parámetro es específico de la arquitectura GAT, donde se han configurado 4 cabezales de atención. Así, el modelo aprende abstracciones en subespacios de dimensión $d/h = 16$.
- `DROPOUT`: Se ha fijado una tasa de 0.5 para las capas de GCN y GAT. Debido a la alta tendencia de las redes de grafos a sobreajustarse durante el entrenamiento, un valor de *dropout* más elevado que el usado en los modelos secuenciales resulta indispensable para forzar al modelo a extraer patrones más genéricos.

3.6.4. Modelos híbridos con grafos

Las redes de grafos son capaces de modelar relaciones estructurales (estáticas) entre usuarios y películas en todo el ecosistema de datos. Sin embargo, carecen de una noción cronológica. Por el otro lado, los modelos secuenciales como el Transformer captan la evolución de los gustos en el tiempo, pero pueden ignorar similitudes latentes a nivel global. Para unificar ambas fortalezas, se ha propuesto una arquitectura híbrida que integra los *embeddings* extraídos previamente por una red de grafos (GCN, LightGCN o GAT) dentro de un esquema de red neuronal secuencial (como el Transformer, Mamba o xLSTM). Este proceso de fusión se ha dividido en dos mecanismos principales: la fusión temprana de ítems y la fusión tardía de usuario.

Fusión temprana de ítems

En esta primera etapa, el objetivo es enriquecer la representación inicial de cada película en la secuencia antes de ser procesada por el núcleo del modelo secuencial (por ejemplo, el mecanismo de atención multicabezal). Para ello, cada ítem de la secuencia s_t dispone de dos representaciones iniciales distintas:

- Un *embedding* entrenable, $\boldsymbol{e}_{\text{train}} \in \mathbb{R}^d$. Es el equivalente al *embedding* de ítem en los modelos secuenciales.
- Un *embedding* de grafo congelado, $\boldsymbol{e}_{\text{GNN}} \in \mathbb{R}^{d_{\text{GNN}}}$, pre-entrenado mediante una GNN. Que el *embedding* esté congelado implica que no se actualizará durante el entrenamiento.

Antes de combinarlos, se somete al *embedding* congelado de grafo a una fase de estabilización y expansión. Primero, se aplica una capa de normalización, seguida de una proyección lineal hacia un subespacio de mayor dimensionalidad (por ejemplo, d). De esta manera, la red neuronal podrá hacer un reparto más equitativo de la atención prestada a cada *embedding*:

$$\tilde{\boldsymbol{e}}_{\text{GNN}} = \text{LayerNorm}(\boldsymbol{e}_{\text{GNN}})\mathbf{W}_{up} + \boldsymbol{b}_{up},$$

donde $\mathbf{W}_{up} \in \mathbb{R}^{d_{\text{GNN}} \times d}$ y $\boldsymbol{b}_{up} \in \mathbb{R}^d$ son parámetros aprendibles. A continuación, se combinan ambas representaciones,

$$\boldsymbol{c}_t = \text{Concat}\left(\boldsymbol{e}_{\text{train}}, \tilde{\boldsymbol{e}}_{\text{GNN}}\right),$$

donde Concat denota la operación de concatenación. El vector resultante $\boldsymbol{c}_t \in \mathbb{R}^{2d}$ transita por dos capas encargadas de reducir la dimensionalidad de una manera gradual hasta recuperar la dimensión original d del modelo secuencial. Esto responde a la necesidad de evitar cuellos de botella representacionales (véase Goodfellow et al., 2016). Contraer de forma abrupta el espacio de características de una sola vez podría provocar pérdidas mayores de información, especialmente al tratar con clases de *embeddings* heterogéneas como es el caso. Así, la primera capa proyecta la concatenación a un espacio intermedio de dimensión $2d \cdot 2/3$, aplicando seguidamente la función de activación ReLU y una capa de *dropout*. La segunda proyección recupera la dimensión d. Se definen de la siguiente manera:

$$\begin{aligned} \boldsymbol{z}_t &= \text{ReLU}(\mathbf{W}_{f1}\boldsymbol{c}_t + \boldsymbol{b}_{f1}), \\ \boldsymbol{x}_t^{(0)} &= \mathbf{W}_{f2}\boldsymbol{z}_t + \boldsymbol{b}_{f2}, \end{aligned}$$

donde $\mathbf{W}_{f1} \in \mathbb{R}^{2d \cdot \frac{2}{3} \times 2d}$, $\boldsymbol{b}_{f1} \in \mathbb{R}^{2d \cdot \frac{2}{3}}$, $\mathbf{W}_{f2} \in \mathbb{R}^{d \times 2d \cdot \frac{2}{3}}$ y $\boldsymbol{b}_{f2} \in \mathbf{R}^d$ son parámetros de pesos aprendibles. Esta nueva representación híbrida $\boldsymbol{x}_t^{(0)}$ es la que sustituye a los vectores de entrada descritos en secciones anteriores (véanse las Secciones 3.2, 3.3 y 3.4).

Fusión tardía de usuario

Una vez que la secuencia de *embeddings* híbridos ha atravesado los L bloques del modelo correspondiente (ya sea el mecanismo de atención del Transformer o los bloques de Mamba/xLSTM), se obtiene la representación final del modelo, denotada como $\mathbf{H}^{(L)} \in \mathbb{R}^{N \times d}$. Esta matriz contiene el historial temporal del usuario enriquecido con la topología de los ítems, pero carece de información sobre el perfil global del usuario que está realizando la sesión.

Por ello, se incorpora una segunda vía de información proveniente de la red de grafos: el *embedding* estructural del usuario. Sea $\boldsymbol{u}_{\text{GNN}} \in \mathbb{R}^{d_{\text{GNN}}}$ este vector pre-entrenado y congelado. En primer lugar, se proyecta linealmente hacia la dimensión del modelo secuencial d:

$$\tilde{\boldsymbol{u}} = \boldsymbol{u}_{\text{GNN}}\mathbf{W}_u + \boldsymbol{b}_u \in \mathbb{R}^d,$$

donde $\mathbf{W}_u \in \mathbb{R}^{d_{\text{GNN}} \times d}$ y $\boldsymbol{b}_u \in \mathbb{R}^d$ son parámetros entrenables. Para poder operar con la matriz secuencial $\mathbf{H}^{(L)}$, este vector de usuario se expande a lo largo de toda la dimensión temporal, generando una matriz $\mathbf{U} \in \mathbb{R}^{N \times d}$ donde cada fila es una copia idéntica de $\tilde{\boldsymbol{u}}$.

La integración de la representación secuencial con el perfil de usuario no se realiza mediante una simple suma o concatenación, sino a través de un mecanismo de *gating* (puerta de control). Esto permite al modelo decidir para cada instante de tiempo t y para cada dimensión latente, si debe prestar más atención al contexto secuencial o al perfil del usuario. Para calcular la matriz de activación de la puerta $\mathbf{G} \in \mathbb{R}^{N \times d}$, se concatenan ambas representaciones y se aplican una transformación lineal seguida de una función de activación sigmoide (véase la Ecuación 1.1):

$$\mathbf{G} = \sigma\left(\text{Concat}\left(\mathbf{H}^{(L)}, \mathbf{U}\right)\mathbf{W}_{gate} + \boldsymbol{b}_{gate}\right),$$

siendo $\mathbf{W}_{gate} \in \mathbb{R}^{2d \times d}$ la matriz de pesos que evalúa la concatenación, y $\boldsymbol{b}_{gate} \in \mathbb{R}^d$ el vector de sesgos. Finalmente, la salida final $\mathbf{H}_{fused}$ se obtiene mediante una ponderación controlada por $\mathbf{G}$:

$$\mathbf{H}_{fused} = \mathbf{G} \odot \mathbf{H}^{(L)} + (\mathbf{1} - \mathbf{G}) \odot \mathbf{U},$$

donde $\odot$ denota el producto elemento a elemento. Un valor cercano a 1 en $\mathbf{G}$ indicará que la red confía en su predicción basándose casi en su totalidad en las relaciones secuenciales, mientras que un valor cercano a 0 dará prioridad al *embedding* estático del usuario obtenido por la red de grafos. Por último, se extrae el *embedding* correspondiente al último ítem y se pasa por una capa SoftMax para ofrecer la recomendación al usuario.

Capítulo 4

Metodología de entrenamiento y evaluación

El trabajo ha sido realizado completamente en Python, con el apoyo fundamental de la librería `PyTorch` (véase Paszke et al., 2019) para la construcción y entrenamiento de las redes neuronales, la librería `transformers` (véase Wolf et al., 2020) para las arquitecturas homónimas, y la librería oficial `mamba-ssm` (véase Gu y Dao, 2024) para la integración de los bloques Mamba. Asimismo, la extracción de los *embeddings* semánticos de los ítems se ha realizado aprovechando los modelos pre-entrenados disponibles en el biblioteca online de modelos Hugging Face (véanse Wang et al., 2020 y Warner et al., 2025). Siguiendo lo estudiado en el Capítulo 2, la evaluación se ejecuta sobre dos variantes del conjunto de datos:

1. **Secuencias originales:** El conjunto de datos completo que refleja las interacciones naturales de los usuarios, propenso a los sesgos expuestos con anterioridad.

2. **Secuencias truncadas:** Para obtener métricas más cercanas a lo que podría conseguir cada arquitectura en un escenario real. Además, nos permite comparar si el comportamiento de los modelos se conserva al entrenarse sobre secuencias de distinta naturaleza.

4.1. Entrenamiento de los modelos

Inicialmente se adoptó el esquema clásico del aprendizaje automático, estableciendo una partición de los datos de un 80 % para entrenamiento, 10 % para validación y 10 % para test, permitiendo obtener entrenamientos "rápidos" durante la fase de ajuste de hiperparámetros de cada modelo. Sin embargo, con el fin de obtener métricas más consistentes y asegurar que los resultados no dependieran de una partición aleatoria favorable, la evaluación final se ha realizado mediante un proceso de validación cruzada de k iteraciones, con $k = 10$. En cada una de las 10 iteraciones, se ha modificado la partición de los datos, reportando finalmente la media y la desviación típica de las métricas obtenidas sobre el conjunto de test.

Debido a las necesidades computacionales de los modelos de aprendizaje profundo, estos han sido entrenados utilizando una unidad de procesamiento gráfico NVIDIA Tesla T4, con 16 GB de VRAM. Con el objetivo de garantizar una comparativa honesta de los resultados, se han empleado estos mismos recursos en todos los modelos, desde el más simple hasta el más complejo. Para las redes neuronales se ha utilizado el optimizador `AdamW` (véase Loshchilov y Hutter, 2019), con una tasa de aprendizaje inicial de $\eta = 10^{-4}$ y una lógica de parada temprana (*early stopping*) basado en la función de pérdida del conjunto de validación, para prevenir el sobreajuste (véase la Figura 4.1). En concreto, se exige una mejora relativa mínima respecto al mejor resultado histórico, definida por el hiperparámetro $\Delta_{rel} = 0.01$.

El entrenamiento solo se considerará productivo si la pérdida de validación actual ($\mathcal{L}_{val,t}$) satisface la condición:

$$\mathcal{L}_{val,t} < \mathcal{L}_{val,best} \cdot (1 - \Delta_{rel}).$$

En caso de no cumplirse esta desigualdad durante un número consecutivo de épocas superior a la *paciencia* (por ejemplo, $P = 5$), el proceso se interrumpe automáticamente. De esta manera no solo evitamos un sobreajuste de los datos de entrenamiento, sino que garantizamos que el aprendizaje finalice automáticamente una vez se alcanza una cierta convergencia.

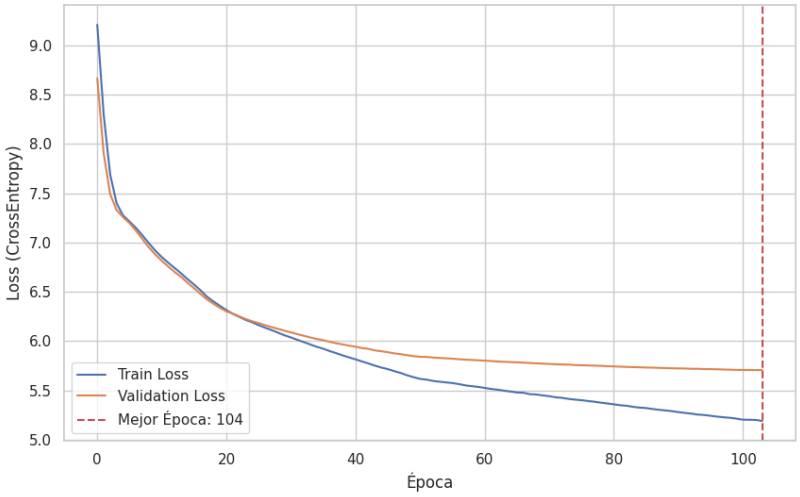

Figura 4.1: Curvas de entrenamiento y validación del modelo Transformer. En el eje horizontal, las iteraciones durante el aprendizaje. En el eje vertical el valor promedio de la función de pérdida $\mathcal{L}_{CE}$.

4.2. Métricas de evaluación

Considérense U el conjunto de usuarios de test y sea R_u^K el conjunto ordenado de las K recomendaciones con mayor probabilidad asignada para cada usuario u. Para un usuario con una secuencia de $n < N$ ítems (recordando que N es el número máximo de secuencia), se toma la secuencia $(s_1, \ldots, s_{n-1})$ y se evalúa el rendimiento del modelo al predecir el último ítem o NBI, s_n^u. Así, para medir la calidad de las recomendaciones, se han seleccionado tres métricas reconocidas en la literatura de sistemas de recomendación:

- ***Hit rate* (HR@K):** Mide la proporción de casos en los que el ítem objetivo se encuentra dentro de la lista de las K mejores recomendaciones del modelo.

$$\text{HR@K} = \frac{1}{|U|} \sum_{u \in U} \mathbb{1}(s_n^u \in R_u^K). \tag{4.1}$$

- ***Mean reciprocal rank* (MRR@K):** Penaliza las predicciones en función de la posición que ocupa el ítem correcto dentro de la lista de recomendaciones.

$$\text{MRR@K} = \frac{1}{|U|} \sum_{u \in U} \frac{1}{\text{rank}(s_n^u, R_u^K)}, \tag{4.2}$$

  donde $\text{rank}(x, L)$ mide la posición de x en la lista ordenada L. Si el ítem no está en el top K, el cociente dentro del sumatorio toma el valor 0.

- ***Normalized discounted cumulative gain* (NDCG@K):** Evalúa la calidad de la recomendación otorgando una importancia logarítmica a la posición del acierto. La formulación original propuesta por Järvelin y Kekäläinen, 2002 es la siguiente:

$$\text{NDCG@K} = \frac{1}{|U|} \sum_{u \in U} \frac{\text{DCG@K}^u}{\text{IDCG@K}^u}, \tag{4.3}$$

donde el DCG@K

$$\text{DCG@K}^u = \sum_{i=1}^{K} \frac{2^{rel_i} - 1}{\log_2(i+1)}, \tag{4.4}$$

siendo rel_i un indicador de relevancia para el ítem en la posición i, e IDCG@K^u el máximo valor posible de DCG@K^u. La tarea de predicción del NBI nos lleva a definir rel_i como un binario, valiendo 1 si el ítem está en las K primeras recomendaciones y 0 en el caso contrario. Por otro lado, el valor máximo que puede tomar DCG@K^u es 1 (el caso ideal donde el NBI está en la primera posición). Así pues, la métrica final resulta un MRR@K que castiga algo menos el hecho de predecir en posiciones inferiores.

$$\text{NDCG@K} = \frac{1}{|U|} \sum_{u \in U} \frac{1}{\log_2(\text{rank}(s_n^u, R_u^K) + 1)} \tag{4.5}$$

Se han tomado valores de $K \in \{1, 20, 100, 300\}$ para las tres métricas, y seleccionado $K = 20$ como el de referencia principal para un caso real. Al fin y al cabo, representa un cantidad de opciones realista que un usuario promedio exploraría en la interfaz de una plataforma de *streaming* o similar. Además de la precisión, se monitorizarán otras dos métricas adicionales:

- **Tiempo de entrenamiento:** representa el coste computacional requerido para que el modelo alcance una cierta convergencia en la función de pérdida. Es crítico para evaluar la escalabilidad de los modelos en un entorno de producción real, donde a la hora de escoger un modelo puede primar más la velocidad que las propias métricas de precisión. Mientras que para las arquitecturas basadas en redes neuronales se estudia este tiempo de entrenamiento, para los algoritmos clásicos como el Item-KNN (que carecen de esta fase) se medirá estrictamente el **tiempo de computación**.
- **Personalización:** con el fin de analizar la capacidad de personalización de cada modelo. La popularidad de un ítem se mide como el número de veces que aparece en el conjunto de datos. Así, el conjunto de datos tendrá una cierta distribución de popularidad (véase la Figura 4.2) con la que contrastar las predicciones de cada modelo. Un modelo con baja capacidad de personalización tenderá a predecir taquillazos.

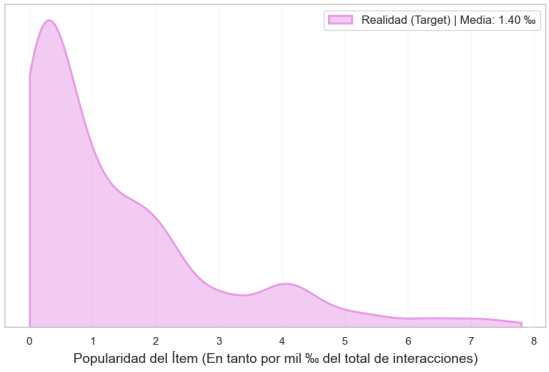

Figura 4.2: Estimación tipo núcleo de la densidad de la distribución de popularidad para las películas en el conjunto de secuencias de MovieLens.

Capítulo 5

Resultados y discusión

En este capítulo se estudiarán los resultados obtenidos al entrenar los siguientes modelos, con sus hiperparámetros debidamente optimizados:

- **Métodos clásicos:** Item-KNN, debido a su mejor desempeño en relación a otros algoritmos clásicos como User-KNN o SVD (véase la Subsección A.1.1).
- **Modelos secuenciales:** Transformer, Mamba y xLSTM. Estos presentan importantes mejoras con respecto a los métodos tradicionales.
- **Redes de grafos:** GCN, LightGCN y GAT. Aunque su mayor utilidad vendrá de ser usados como arquitectura complementaria en los Modelos Híbridos, las redes de grafos pueden ser utilizadas por sí mismas como sistemas de recomendación (véase la Sección 3.6).
- **Modelos híbridos:** Transformer+BERT, Transformer+GAT y xLSTM+GAT. Por una cuestión de tiempos de entrenamiento, se han selecionado estas tres arquitecturas en detrimento de otras combinaciones posibles como podrían haber sido Mamba+LightGCN o xLSTM+BERT.

Además, se mostrará la puesta en marcha de estos modelos sobre una plataforma interactiva, simulando un escenario de recomendación real. Por último, se realizará una reflexión concluyente sobre el trabajo realizado.

5.1. Resultados de la experimentación

En primer lugar, se muestran las métricas relativas a los modelos comentados, para las secuencias originales (véase la Tabla 5.1) y para las secuencias truncadas (véase la Tabla 5.2). Para una comparativa global, se han seleccionado las siguientes métricas como las más representativas: HR@1 para medir la precisión al predecir un único ítem, HR@20 para evaluar los modelos en un escenario real, HR@300 para analizar la robustez del modelo en horizontes amplios y NDCG@20 para examinar la calidad posicional del *ranking* de recomendaciones.

Por otro lado, la métrica de personalización se ha representado en tres niveles: bajo (*), que indica una tendencia del modelo a recomendar casi exclusivamente los ítems más populares o "taquillazos"; medio (**), donde se observa un balance razonable entre recomendaciones globales y contenido específico; y alto (***), que refleja una gran capacidad de la arquitectura para adaptarse a los gustos individuales y recomendar ítems "nicho". Los resultados completos de cada modelo, con medias y desviaciones típicas para cada métrica, se pueden consultar en el Apéndice A.

Modelo	HR@1	HR@20	HR@300	NDCG@20	Pers.	Tiempo
Item-KNN	1.99 %	21.9 %	77.1 %	0.1489	**	**0.5 s**
Transformer	8.16 %	32.6 %	74.8 %	0.1753	***	8.60 min
Mamba	6.70 %	29.0 %	69.2 %	0.1526	***	3.22 min
xLSTM	7.83 %	31.1 %	72.3 %	0.1682	***	7.32 min
GCN	1.40 %	13.2 %	52.2 %	0.0552	*	4.97 s
LightGCN	1.85 %	14.5 %	54.2 %	0.0633	*	4.78 s
GAT	1.89 %	18.1 %	67.8 %	0.0744	**	5.5 min
Tr.+BERT	7.66 %	34.3 %	76.2 %	0.1788	***	7.33 min
Tr.+GAT	8.54 %	34.8 %	**77.4 %**	0.1870	***	9.31 min
xLSTM+GAT	**8.98 %**	**36.1 %**	76.0 %	**0.1945**	***	15.4 min

Tabla 5.1: Resultados globales sobre el conjunto de secuencias originales. Los valores en negrita indican el mejor rendimiento por métrica.

Modelo	HR@1	HR@20	HR@300	NDCG@20	Pers.	Tiempo
Item-KNN	3.07 %	21.8 %	**73.9 %**	0.0953	**	**0.3 s**
Transformer	5.30 %	24.0 %	69.8 %	0.1231	***	6.11 min
Mamba	4.11 %	20.3 %	62.9 %	0.1003	***	2.6 min
xLSTM	4.83 %	23.6 %	67.7 %	0.1189	***	5.68 min
GCN	0.80 %	8.3 %	47.3 %	0.0340	*	3.46 s
LightGCN	0.83 %	9.6 %	46.5 %	0.0373	*	3.39 s
GAT	1.07 %	14.6 %	65.1 %	0.0556	**	4.39 min
Tr.+BERT	4.91 %	25.9 %	72.1 %	0.1273	***	5.56 min
Tr.+GAT	5.69 %	25.8 %	72.5 %	0.1319	***	6.26 min
xLSTM+GAT	**6.17 %**	**27.5 %**	71.6 %	**0.1418**	***	12.1 min

Tabla 5.2: Resultados globales sobre el conjunto de secuencias truncadas por la izquierda. Los valores en negrita indican el mejor rendimiento por métrica.

5.1.1. Métricas de NBI

Los resultados de la Tabla 5.1 demuestran que los modelos secuenciales (Transformer, Mamba y xLSTM) proporcionan un salto de calidad predictiva respecto a los algoritmos clásicos. En general, estos superan ampliamente al Item-KNN, que logra un HR@20 del 21.9 % y un NDCG@20 de 0.1489. En segundo lugar, los modelos basados en grafos (GCN, LightGCN y GAT) no logran un mejor rendimiento, obteniendo peores resultados en todas las métricas. A pesar de ello, el modelo basado en grafos con atención se queda cerca, obteniendo un HR@20 del 18.1 % y un NDCG@20 de 0.0744. Finalmente, se observa una mejora de los modelos híbridos con respecto a las arquitecturas individuales, alcanzando xLSTM+GAT las mejores puntuaciones, con un HR@20 del 36.1 % y un NDCG@20 de 0.1945.

No obstante, el comportamiento de los modelos cambia drásticamente al ser evaluados sobre el conjunto de secuencias truncadas (véase la Tabla 5.2). En este escenario, diseñado para mitigar los posibles sesgos de MovieLens, se observa una caída generalizada en la capacidad predictiva de casi todas las arquitecturas de aprendizaje profundo. El modelo Transformer, por ejemplo, reduce su HR@20 de un 32.6 % a un 24.0 %. Podría ser tentador atribuir esta bajada de métricas a una diferencia en el

tamaño del conjunto de datos: como veíamos en el Capítulo 2, el conjunto de secuencias originales y el conjunto de secuencias truncadas presentan una diferencia del 12 % (10271 a 9121 secuencias). Sin embargo, esto no es justificación para tal descenso en la calidad de las recomendaciones. Como adelantábamos en la Tabla 2.1, reducir o aumentar el tamaño de la muestra (tomando como referencia el 0.3) no repercute tan drásticamente en los resultados finales. Este fenómeno confirma lo estudiado por Fan et al. (2024), quienes afirman que las primeras valoraciones de un usuario definen en gran parte lo que "verá" en un futuro. Frente a esta sensibilidad al truncado de las secuencias, destaca el Item-KNN, pues apenas sufre variaciones en su rendimiento al eliminar el historial inicial.

5.1.2. Capacidad de personalización

Analizar la capacidad de personalización resulta indispensable para complementar las métricas de precisión, pues permite evaluar en qué medida un modelo es capaz de adaptarse a los gustos específicos de un usuario. Es posible consultar todos los gráficos relativos a la capacidad de personalización en la Sección A.2 del Apéndice. Los modelos secuenciales e híbridos muestran los mejores resultados (véase la Figura 5.1), obteniendo distribuciones muy similares a las del conjunto de datos original.

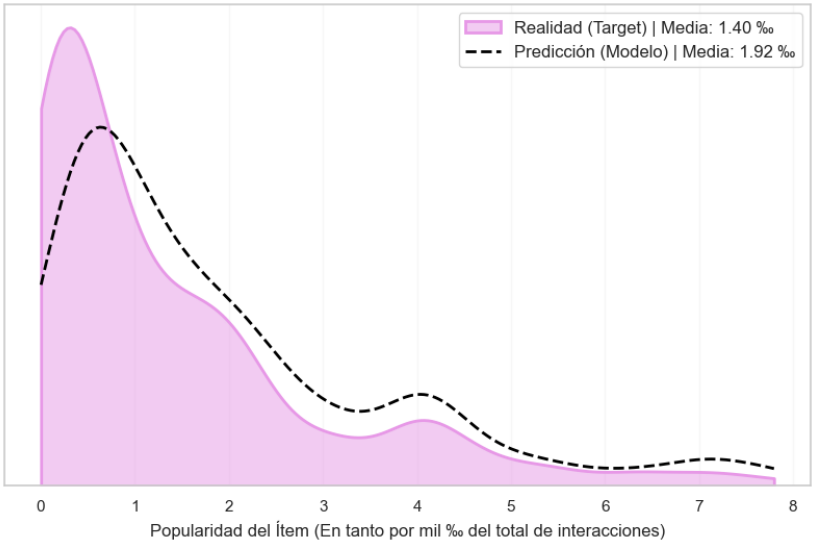

Figura 5.1: Estimación tipo núcleo de la densidad de la distribución de popularidad para las películas en el conjunto de secuencias de MovieLens comparado con la predicción del modelo Transformer. El área sombreada representa el comportamiento real de los usuarios, mientras que la línea discontinua ilustra las recomendaciones del modelo. La leyenda detalla la popularidad media (número medio de apariciones en secuencias) de los ítems, en tanto por mil.

Los modelos basados en grafos puros (GCN y LightGCN) sufren especialmente en este aspecto, limitándose de forma casi exclusiva a recomendar las películas más populares de la plataforma (los "taquillazos"). Como se aprecia en la Figura 5.2, la distribución de las recomendaciones generadas por estas arquitecturas se solapa con la de los ítems de mayor frecuencia en el conjunto de entrenamiento. Al carecer de un mecanismo de ponderación de la importancia de cada ítem o de la secuencialidad, estas redes propagan de forma masiva las representaciones de los nodos con mayor grado (es decir, las películas más vistas). De esta manera se anula cualquier intento de personalización y la red es saturada con contenido genérico.

Por otra parte, arquitecturas como Item-KNN (véase la Figura A.1) o el modelo de grafos con

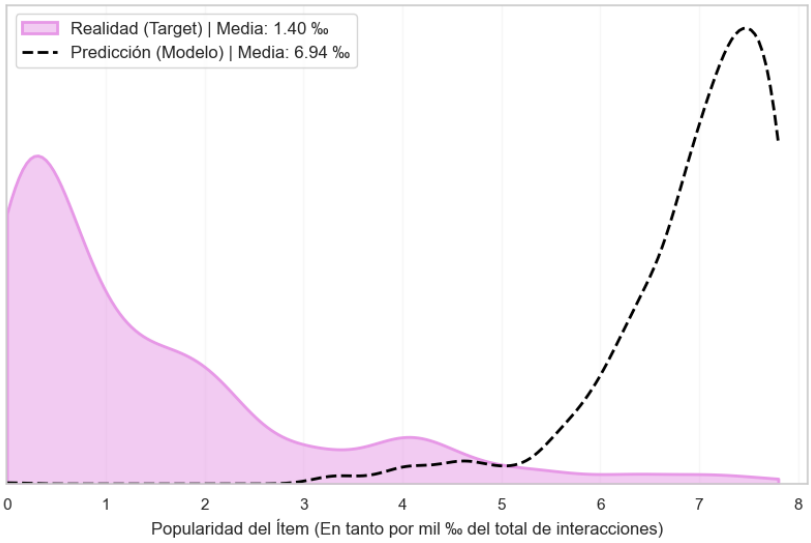

Figura 5.2: Estimación tipo núcleo de la densidad de la distribución de popularidad para las películas en el conjunto de secuencias de MovieLens comparado con la predicción del modelo LightGCN. El área sombreada representa el comportamiento real de los usuarios, mientras que la línea discontinua ilustra las recomendaciones del modelo. La leyenda detalla la popularidad media (número medio de apariciones en secuencias) de los ítems, en tanto por mil.

atención (véase la Figura A.5) presentan un comportamiento mucho más equilibrado. En el caso de GAT, el mecanismo de atención mitiga la influencia del grado del nodo, permitiendo otorgar más importancia a películas "nicho". En el caso del Item-KNN, su naturaleza matricial consigue un efecto similar, fruto de buscar similitudes directas ítem a ítem en lugar de promediar *embeddings*.

5.1.3. Tiempo de entrenamiento

La viabilidad de un sistema de recomendación en entornos de producción reales no depende únicamente de su precisión o capacidad de personalización, sino también de su coste computacional. Las diferencias entre los tiempos de entrenamiento y computación requeridos revelan un fuerte contraste entre la sencillez de los enfoques tradicionales y las exigencias de los modelos basados en redes neuronales profundas.

Como es de esperar, los métodos clásicos como el Item-KNN resultan extremadamente rápidos en comparación con cualquier red neuronal. Al tratarse de un algoritmo de *lazy learning* (véase la Sección 3.1), no requiere una fase de entrenamiento, sino que calcula las relaciones de similitud directamente sobre la matriz de interacciones de manera prácticamente instantánea, dependiendo del tamaño de la matriz. Sin embargo, esta inmediatez tiene como contrapartida un estancamiento en las métricas de rendimiento, lo que justifica la necesidad de explorar modelos más avanzados. Además, entornos con gran cantidad de datos podrían imposibilitar el uso de arquitecturas basadas en la matriz de interacción, por una cuestión de memoria computacional.

En este sentido, las arquitecturas secuenciales profundas logran dar el salto de calidad predictiva comentado anteriormente, pero a costa de un gran incremento en el tiempo de computación. En este caso, la mayoría de entrenamientos duran entre 5 y 15 minutos, pero esto depende en gran parte de la infraestructura utilizada. Por otro lado, un hallazgo relevante de la experimentación es que añadir complejidad estructural no siempre se traduce en un aumento proporcional del tiempo de

entrenamiento. Dar más información al modelo o darle más capacidad de representación (por ejemplo, aumentando la dimensión de los *embeddings*, véase la Tabla 3.2) puede incluso reducir el tiempo requerido por la red neuronal, al precisar menos iteraciones para alcanzar valores similares en la función de pérdida.

Por otro lado, los tiempos del modelo xLSTM no son representativos del mismo, como se mencionó en la Sección 3.4. Debido a restricciones en las capacidades de la GPU utlizada, no fue posible compilar ni ejecutar el paquete optimizado de Python correspondiente. Esto obligó a definir toda la arquitectura de forma manual, careciendo así de la optimización realizada por Beck et al. (2024). A pesar de esto, las métricas de precisión no se ven afectadas por esta diferencia.

Por último, cabe destacar el comportamiento de los grafos con atención. GAT no solo arroja peores métricas de precisión que los modelos secuenciales, sino que además presenta tiempos de entrenamiento más o menos equivalentes. El cálculo de los coeficientes de atención para cada par de nodos conectados en el grafo introduce una carga computacional muy alta en cada iteración. De forma individual, las GNN no resultan competitivas para resolver este problema de NBI, estando su verdadero valor en ser utilizadas como componentes complementarios dentro de un marco híbrido.

5.1.4. Escalabilidad respecto a la longitud del historial

Otro aspecto fundamental a la hora de evaluar la viabilidad de una arquitectura de *sequence modeling* en entornos de producción es su escalabilidad respecto a la longitud del historial del usuario. Modificar el parámetro de longitud máxima de las secuencias (`MAX_LENGTH`) tiene un impacto directo en la cantidad de operaciones requeridas, el uso de memoria VRAM y, por ende, en los tiempos de entrenamiento. Para ilustrarlo, se ha realizado una comparativa entre las arquitecturas Transformer y Mamba bajo las mismas condiciones, variando la longitud máxima de las secuencias, $N \in \{20, 60, 90, 200\}$. Los resultados se detallan en la Tabla 5.3.

N	**Transformer**	**Mamba**
20	2.07 min	1.15 min
60	2.93 min	1.72 min
90	8.60 min	3.22 min
200	29.98 min	8.27 min

Tabla 5.3: Tiempos de entrenamiento requeridos para las arquitecturas Transformer y Mamba en función de la longitud máxima de secuencia N.

La diferencia en los tiempos de ejecución entre ambas arquitecturas se acentúa a medida que las secuencias se alargan. Esto se debe precisamente a la formulación de cada modelo:

- El mecanismo de atención multicabezal introducido en la Sección 3.2 requiere calcular una matriz de atención para cada posición del historial, en la que cada película se compara con todas las demás. Esto implica que la complejidad computacional y de memoria escala de forma cuadrática respecto a la longitud de la secuencia, es decir, $\mathcal{O}(N^2)$. Al pasar de una longitud máxima de 90 ítems a 200, el tiempo de entrenamiento del Transformer sube considerablemente (desde 8 hasta 30 minutos).
- Por el contrario, Mamba procesa la información mediante recurrencia a través del SSM Selectivo, comentado en la Sección 3.3. Al depender de un estado oculto que se actualiza iterativamente, logra un coste computacional estrictamente lineal $\mathcal{O}(N)$. Así, para secuencias de 200 ítems, Mamba apenas requiere 8.27 minutos, resultando una alternativa más eficiente en este contexto.

Cabe mencionar que la arquitectura xLSTM fue también concebida para ofrecer un escalado lineal, pero no se muestra debido a la problemática explicada con anterioridad.

Es importante señalar que el tiempo de entrenamiento empírico está sujeto a múltiples variables además de `MAX_LENGTH`, como puede ser el tamaño de los *embeddings* o los costes fijos de inicialización de la GPU. Así, la escala lineal o cuadrática de los modelos puede quedar enmascarada si se analiza de manera aislada. Por este motivo, resultó indispensable plantear el análisis comparando ambas arquitecturas. Finalmente, y como se puede observar en la Figura 5.3 (meramente ilustrativa), los tamaños de secuencia considerados en este trabajo pueden no ser tan significativos a la hora de computar los modelos. La diferencia sería considerablemente mayor en historiales más extensos.

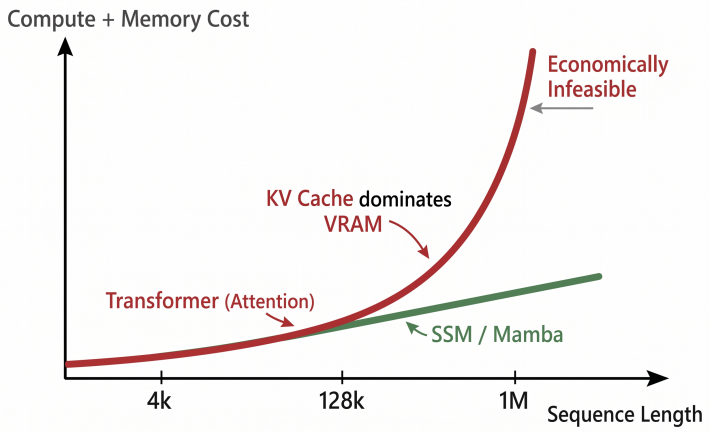

Figura 5.3: Comparativa del escalado de cómputo durante el entrenamiento en función de la longitud de secuencia N para arquitecturas Transformer $\mathcal{O}(N^2)$ y Mamba $\mathcal{O}(N)$. Imagen adaptada de Bansal (2025).

5.2. Puesta en producción de los modelos

Para demostrar de forma práctica los contenidos desarrollados a lo largo de este trabajo, se ha creado una interfaz interactiva emulando el comportamiento del motor de recomendaciones de una plataforma de *streaming*. El desarrollo de esta aplicación web se ha llevado a cabo utilizando `Streamlit` (véase Streamlit Inc., 2026), un paquete de Python diseñado específicamente para el despliegue de aplicaciones de aprendizaje automático. Esta prueba de concepto es plenamente funcional y se puede ejecutar en un entorno local o en la nube.

La aplicación se encuentra estructurada en dos módulos: por un lado, una plataforma de inferencia orientada a ofrecer recomendaciones en tiempo real en base a un historial de visualización. Por otro lado, un visualizador del espacio de *embeddings* de ítems, enfocándose en los modelos usados para las variantes híbridas (BERT y GAT). Estas interfaces permiten evaluar la robustez de las predicciones de una forma más cualitativa que cuantitativa.

Al terminar la fase de entrenamiento de un modelo, los pesos aprendidos por la red neuronal pertinente se almacenan en ficheros con extensión `.pth`. De esta forma, generar recomendaciones para un usuario nuevo es tan sencillo como introducir su historial en esta red con pesos previamente aprendidos. Esta fase de inferencia resulta mucho más veloz que el entrenamiento, lo que hace viable desarrollar esta aplicación.

En primer lugar se encuentra la página *Sequential Playground* (véase la Figura 5.4). Esta permite emitir recomendaciones cinematográficas a usuarios que hayan insertado su historial previamente,

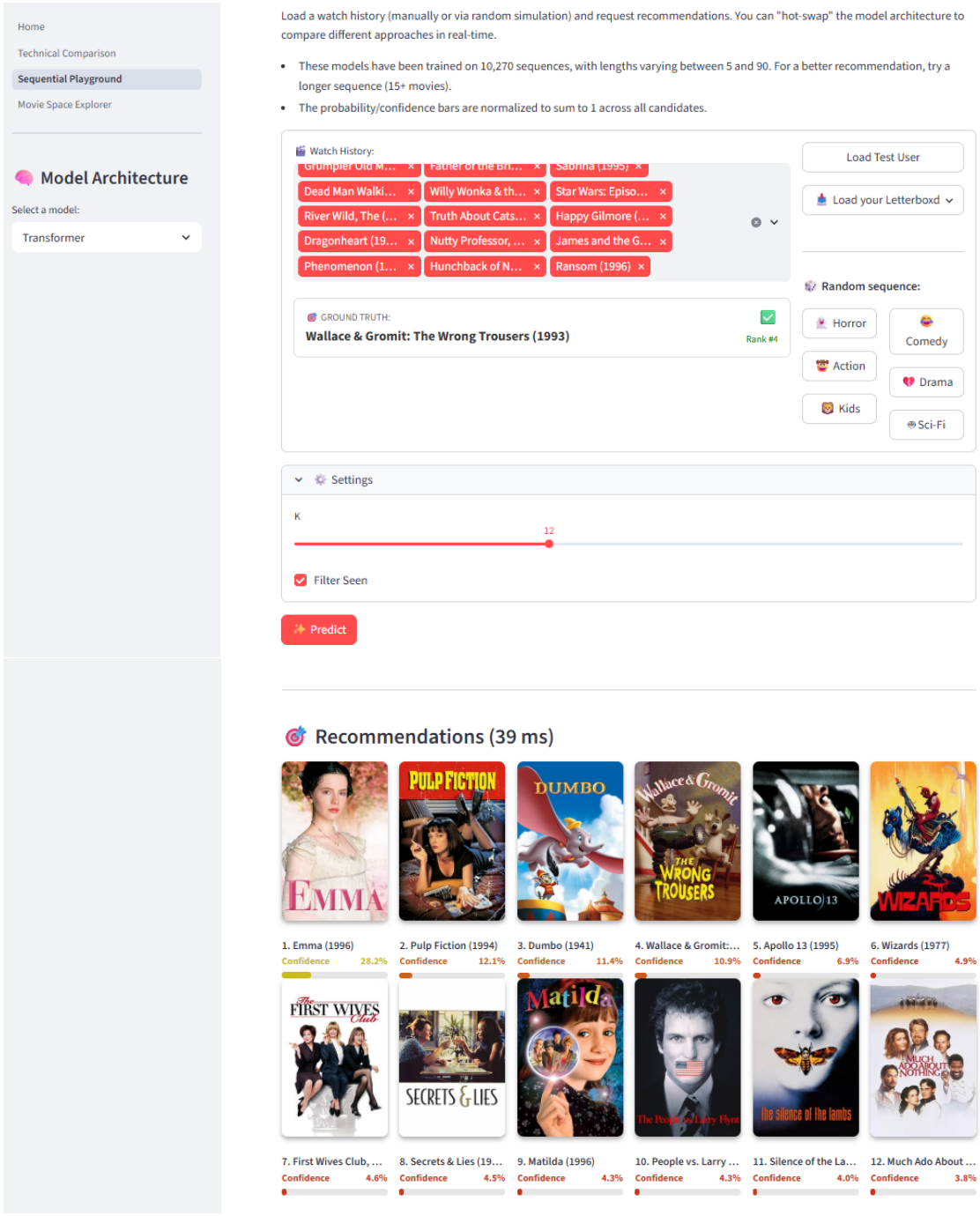

Figura 5.4: Interfaz de la página *Sequential Playground*, habiendo generado $K = 12$ recomendaciones para un usuario de test. En la parte inferior se muestran las películas sugeridas junto a la confianza del modelo. Estos porcentajes representan las probabilidades predictivas obtenidas en la capa de salida (SoftMax), normalizadas sobre estas 12 recomendaciones. En este caso, el modelo Transformer ha acertado la próxima película consumida por el usuario, *Wallace & Gromit: The Wrong Trousers*, en la posición número 4 y otorgándole una confianza del 10.9 %.

observando las diferencias seleccionando distintos modelos. A la hora de recibir un número K establecido de recomendaciones, se puede realizar una evaluación cualitativa de dos formas:

- Validación con usuarios de test: Al cargar la cronología de usuarios de MovieLens en el conjunto de test, es posible observar si el modelo ha sido capaz de predecir correctamente la película objetivo.
- Integración con usuarios reales: Además, la plataforma permite a usuarios con cuenta en la red social Letterboxd (véase Letterboxd Limited, 2011) cargar su historial personal de películas vistas. De esta manera, el cliente puede evaluar la calidad predictiva de las recomendaciones de forma empírica. También es posible simular historiales ficticios restringidos a un género en particular (por ejemplo, terror).

Por otro lado, la página *Movie Space Explorer* (véase la Figura 5.5) permite analizar el funcionamiento de las arquitecturas complementarias usadas en los modelos híbridos: MiniBERT, ModernBERT y GAT. En este apartado se debe seleccionar una película de entre el catálogo. Utilizando la matriz de *embeddings* de cada modelo, el sistema calcula los vecinos más cercanos utilizando la similitud coseno (véase la Ecuación 3.1), construyendo así un grafo tridimensional donde la película seleccionada se encuentra en el nodo central. La disposición de sus K-vecinos se calcula teniendo en cuenta la similitud con la película seleccionada (distancia al centro), y el algoritmo de fuerzas dirigidas de Fruchterman-Reingold, implementado mediante la librería `NetworkX` (Hagberg et al., 2008). La película central permanece anclada en el origen, mientras que las aristas actúan como resortes cuya fuerza de atracción es directamente proporcional a la similitud coseno entre los *embeddings* de las películas conectadas.

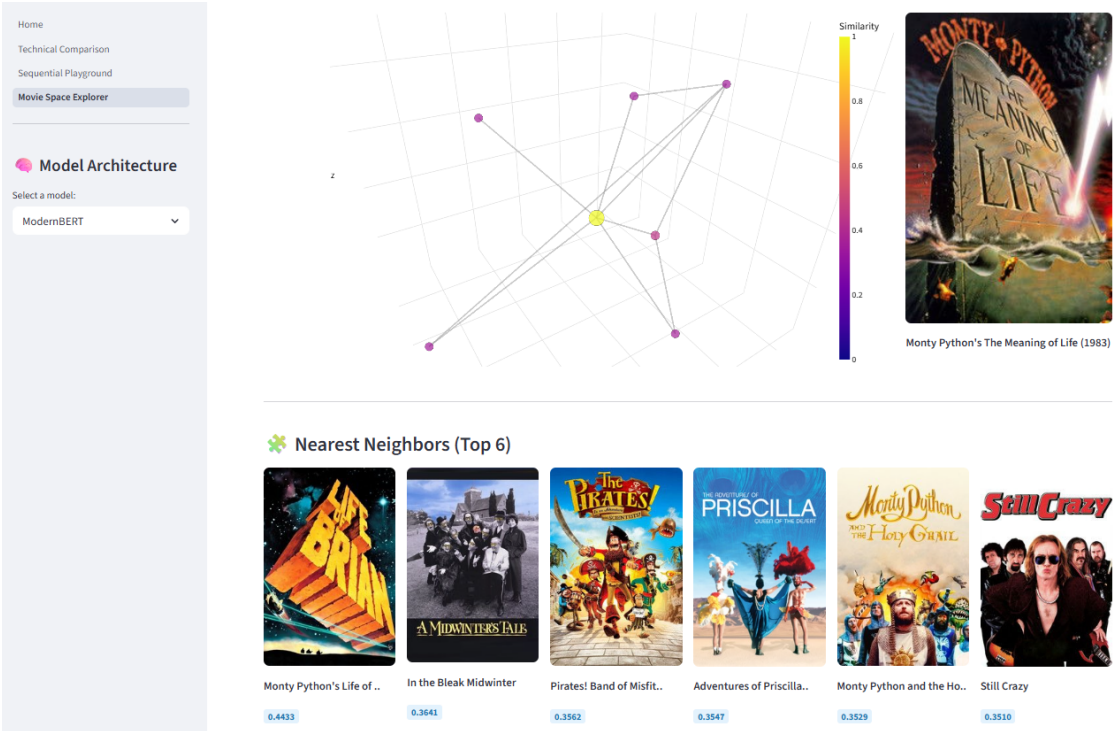

Figura 5.5: Interfaz del módulo *Movie Space Explorer*. Se presenta el grafo correspondiente a la película *Monty Python's The Meaning of Life*, junto con sus K vecinos más cercanos. En la parte inferior se muestran las películas más afines, acompañadas por un valor numérico que representa la similitud coseno entre sus respectivos *embeddings* y el de la película de referencia. Esto permite interpretar la proximidad semántica o colaborativa capturada por el modelo.

El grafo construido tendrá un significado diferente dependiendo de la arquitectura elegida:

- Modelos BERT: Son modelos de contenido, por lo que la similitud estará basada en los títulos, sinopsis, etc (véase la Sección 3.5). Vecinos cercanos a *Spider-Man* podrían ser *Spider-Man II* o *Spider-Man: Into the Spider-Verse*.
- GAT: Se trata de un modelo colaborativo. Así pues, una película será muy similar a otra si ambas han sido consumidas por un mayor número de usuarios en común. Vecinos cercanos a *Spider-Man* podrían ser *The Avengers* o *Fantastic Four*.

Esta página ha resultado especialmente útil para resaltar las diferencias entre los modelos de lenguaje MiniBERT y ModernBERT. El primero tiende a agrupar películas basándose en una coincidencia más superficial, por palabras clave o fraseología concreta. Por el contrario, ModernBERT demuestra una mayor capacidad representacional, logrando agrupar películas conceptualmente afines (ya sea por narrativa o temática) incluso cuando hay disparidad en sus metadatos.

5.2.1. Posibles desafíos

Una de las principales lecciones extraídas de esta trabajo es la existencia de una brecha evidente entre la evaluación teórica y el impacto empresarial real. Los conjuntos de datos académicos como MovieLens no representan al completo la complejidad de entornos de producción, puesto que suelen presentar sesgos como los ya vistos en el Capítulo 2. Como consecuencia, los resultados presentados pueden pecar de optimismo, y las mejoras en métricas como HR@K o NDCG@K no se traducen necesariamente en un incremento directo de las ventas o del nivel de compromiso (*engagement*) del cliente.

En segundo lugar, el problema del arranque en frío sobresale en los sistemas de recomendación industriales. Cuando un nuevo usuario accede a la plataforma sin un historial de transacciones registrado, resulta complejo aplicar modelos secuenciales. Por otro lado, la introducción de nuevos ítems en el catálogo exige la incorporación de mecanismos de contingencia, tales como la recomendación de ítems "tendencia" o la extracción de *embeddings* semánticos mediante BERT. La naturaleza dinámica y rotativa de los catálogos de ítems en entornos, como el comercio electrónico o los portales de noticias, suponen otro tipo de retos. En estos casos se suele recurrir a estrategias de reentrenamiento periódicas (por ejemplo, cada noche) o a priorizar los ítems más recientes mediante ponderaciones.

Finalmente, el comportamiento registrado en las cuentas de usuario de entornos reales como las plataformas de *streaming* es inherentemente multiperfil, puesto que una misma cuenta suele compartirse entre varios individuos o núcleos familiares. Ante esta mezcla de comportamientos, un modelado basado en sesiones podría ser más eficaz que el basado estrictamente en perfiles estáticos de usuario. Este tipo de modelado podría aprovecharse de señales externas como la hora del día, el tipo de dispositivo o un identificador único de sesión. De esta manera, se podría definir el perfil que se encuentra activo en cada instante de tiempo t, mejorando la capacidad predictiva.

5.3. Conclusiones

La evaluación y comparación de las distintas arquitecturas planteadas en este trabajo ha permitido contrastar enfoques que abarcan desde el filtrado colaborativo tradicional hasta las redes neuronales secuenciales más modernas. Esto ha puesto de manifiesto un principio fundamental en el diseño de sistemas de recomendación: no existe una solución universal. Al final, la elección de un modelo está sujeta a un balance entre el rendimiento predictivo y las restricciones operacionales.

Se ha mostrado que el incremento en la complejidad estructural de las redes se traduce directamente en mejoras en la predicción del NBI, y en una mayor capacidad de personalización. Sin embargo, esto acarrea penalizaciones en términos de consumo de memoria y tiempos de entrenamiento, además de sufrir especialmente ante problemas como el ya mencionado arranque en frío. Con el fin de facilitar una visión global, en la Tabla 5.4 se presenta una síntesis de las principales fortalezas y debilidades identificadas para cada arquitectura estudiada.

Modelo	Fortalezas	Debilidades
Item-KNN	Inferencia instantánea (*lazy learning*), interpretable y altamente robusto al truncado de secuencias.	Arquitectura estática (ignora la cronología). Alta demanda de memoria para matrices masivas.
Transformer	Excelente precisión predictiva y alta capacidad de personalización.	Coste computacional y de memoria cuadrático $\mathcal{O}(N^2)$. Susceptible al problema de arranque en frío.
Mamba	Escalabilidad lineal $\mathcal{O}(N)$ ideal para historiales extensos. Alta robustez ante restricciones de infraestructura.	Precisión NBI ligeramente inferior al Transformer en secuencias de longitud moderada.
xLSTM	Precisión superior al Transformer y escalabilidad lineal $\mathcal{O}(N)$.	Falta de soporte optimizado en librerías actuales para ciertas GPUs básicas, limitando su velocidad práctica.
BERT	Aportan contexto semántico a partir de metadatos. ModernBERT logra capturar similitudes más profundas que MiniBERT.	Exigen una intensa fase de cómputo para generar los *embeddings* antes de ser usados para la red principal.
GCN / LightGCN	Alta velocidad de cómputo, simplicidad algorítmica y mayor interpretabilidad estructural.	Precisión predictiva deficiente y nula personalización (sesgo de popularidad). Incapaces de procesar nuevos usuarios en inferencia.
GAT	Mejora notable en métricas de NBI frente a las GCN. Alternativa viable para secuencias cortas.	El cálculo de atención eleva los tiempos de entrenamiento. Requiere que los usuarios evaluados existan en el grafo inicial.
Modelos híbridos	Rendimiento máximo en precisión. Mitigan el problema del arranque en frío aportando contexto inicial (semántico o colaborativo).	Incremento drástico de la complejidad y uso de memoria. Exigen fases previas adicionales de entrenamiento o extracción de características.

Tabla 5.4: Resumen comparativo de las principales ventajas y desventajas de las arquitecturas evaluadas.

5.3.1. Investigación futura

A la luz de las conclusiones obtenidas en este trabajo, surgen varias propuestas de investigación. En primer lugar, se plantea un modelo avanzado basado en valoraciones. Esta metodología consistiría en concatenar las puntuaciones numéricas otorgadas por los usuarios a cada ítem directamente dentro de los vectores de la secuencia cronológica, justo antes de alimentar a la red neuronal oportuna. Al integrar este contexto, el modelo no solo procesaría la trayectoria cronológica del usuario, sino también su nivel de satisfacción en cada instante t, lo que podría significar un avance en cuestión de métricas.

En segundo lugar, se propone el diseño de una nueva arquitectura híbrida enfocada en la computación

en tiempo real. Esta combinaría un modelo secuencial como los ya presentados, con la capacidad colaborativa del algoritmo Item-KNN. Al ser un método de *lazy learning*, la parte de inferencia pertinente implicaría un ligero aumento en los tiempos de computación. A pesar de ello, es posible que este problema se pueda mitigar, buscando además una mejora en las métricas de precisión (ya que obtuvo mejores resultados individuales que las redes de grafos).

Asimismo, una línea de investigación prometedora consiste en arquitecturas compuestas mediante la combinación o el apilamiento de bloques provenientes de distintos modelos secuenciales. Como muestran Lieber et al. (2024), integrar capas basadas en el mecanismo de atención del Transformer junto con bloques SSM de Mamba resulta en modelos con una capacidad de representación superior. Por otro lado, el artículo original de xLSTM (véase Beck et al., 2024) ya sugería este concepto de hibridación de arquitecturas secuenciales, ya que estas serían capaces de asimilar relaciones contextuales globales (Transformer), y al mismo tiempo, estructuras recurrentes de memoria (xLSTM).

Por último, con el objetivo de valorar los modelos con una mayor robustez, sería interesante evaluar las arquitecturas consideradas en esta memoria utilizando otros conjuntos de datos, de naturaleza similar o radicalmente distinta. Además, mercería la pena explorar longitudes de secuencia sustancialmente mayores a las empleadas en este trabajo. Esto permitiría comprobar de forma empírica si las ventajas del escalado lineal $\mathcal{O}(N)$ de Mamba y xLSTM frente al coste cuadrático del Transformer se traducen en una eficiencia extremadamente determinante en esos casos.

5.3.2. Conclusiones finales

A lo largo de este Trabajo Fin de Máster se ha explorado y evaluado la intersección entre las metodologías de *next best action* y *sequence modeling*. Los resultados obtenidos justifican el salto necesario desde los algoritmos estáticos tradicionales hacia las redes neuronales secuenciales. Modelos como el Transformer, Mamba y xLSTM han demostrado superioridad frente al filtrado colaborativo clásico (Item-KNN) en términos de precisión predictiva y capacidad de personalización.

Paralelamente, se ha evidenciado la importancia de un buen análisis previo de los datos. El contraste entre los resultados obtenidos sobre el conjunto original de MovieLens y su versión truncada mostró la estructura subyacente al *dataset*, revelando una dependencia que posteriormente influyó en el entrenamiento de las redes neuronales.

Sin duda, el mayor hito de este proyecto reside en el desarrollo de los modelos híbridos. Se han combinado exitosamente arquitecturas secuenciales con información semántica proveniente de modelos de lenguaje BERT, y con información colaborativa a través de redes neuronales de grafos. Estos nuevos modelos han producido los mejores resultados del estudio, culminando en la arquitectura xLSTM+GAT.

Como reflexión final, se ha podido observar que el rendimiento bruto no lo es todo, pues a menudo se precisa un equilibrio entre la precisión del modelo y su viabilidad computacional. Mientras que las arquitecturas más complejas logran unas métricas superiores, métodos más simples o escalables pueden convertirse en alternativas robustas ante distintas problemáticas (por ejemplo, cuando se trabaja con historiales de usuario muy extensos). En definitiva, no existe un modelo universalmente perfecto, sino que la elección de la arquitectura óptima dependerá intrínsecamente de la naturaleza de los datos y de la infraestructura disponible.

Apéndice A

Anexo de experimentación

En este anexo se presentan de forma detallada los resultados obtenidos durante la fase de experimentación y evaluación de las distintas arquitecturas analizadas en el trabajo.

A.1. Métricas

Para cada arquitectura estudiada en el Capítulo 3, se reflejan los resultados numéricos de las métricas HR@K, MRR@K, NDCG@K y el tiempo de entrenamiento (o de computación, si nos referimos a los modelos clásicos), obtenidas por validación cruzada de $k = 10$ iteraciones. Se presentan medias y desviaciones típicas de cada métrica, para cada conjunto de datos (secuencias originales y truncadas por la izquierda).

A.1.1. Métodos clásicos

En este apartado se recogen los resultados de los algoritmos tradicionales de recomendación evaluados. Concretamente, se exponen las métricas para el modelo basado en contenido TF-IDF (Tabla A.1), el algoritmo colaborativo Item-KNN (Tabla A.2), y sus variantes User-KNN (Tabla A.3) y SVD (Tabla A.4).

Métrica	**K=1**	**K=20**	**K=100**	**K=300**
Secuencias Originales				
HR@K (%)	0.48 ± 0.05	3.02 ± 0.21	7.79 ± 0.45	15.82 ± 0.88
MRR@K	0.0048 ± 0.0005	0.0114 ± 0.0010	0.0125 ± 0.0011	0.0129 ± 0.0012
NDCG@K	0.0048 ± 0.0005	0.0157 ± 0.0014	0.0241 ± 0.0018	0.0349 ± 0.0022
			Tiempo de ejecución: 1.21 s ± 0.08 s	
Secuencias Truncadas				
HR@K (%)	1.20 ± 0.12	3.61 ± 0.28	7.78 ± 0.52	15.22 ± 0.91
MRR@K	0.0120 ± 0.0012	0.0181 ± 0.0015	0.0190 ± 0.0016	0.0194 ± 0.0016
NDCG@K	0.0120 ± 0.0012	0.0223 ± 0.0019	0.0296 ± 0.0021	0.0395 ± 0.0025
			Tiempo de ejecución: 1.10 s ± 0.06 s	

Tabla A.1: Resultados detallados (media ± desviación típica) de la validación cruzada para el modelo Content-Based (TF-IDF) sobre ambos conjuntos de datos.

Métrica	**K=1**	**K=20**	**K=100**	**K=300**
Secuencias Originales				
HR@K (%)	1.99 ± 0.18	21.86 ± 1.05	55.56 ± 1.42	77.11 ± 1.15
MRR@K	0.0199 ± 0.0018	0.0522 ± 0.0035	0.0600 ± 0.0038	0.0614 ± 0.0039
NDCG@K	0.0199 ± 0.0018	0.0879 ± 0.0042	0.1489 ± 0.0051	0.1781 ± 0.0048
			Tiempo de ejecución: 0.49 s ± 0.03 s	
Secuencias Truncadas				
HR@K (%)	3.07 ± 0.25	21.80 ± 1.12	51.37 ± 1.38	73.93 ± 1.21
MRR@K	0.0307 ± 0.0025	0.0616 ± 0.0038	0.0681 ± 0.0041	0.0695 ± 0.0042
NDCG@K	0.0307 ± 0.0025	0.0953 ± 0.0049	0.1481 ± 0.0053	0.1784 ± 0.0050
			Tiempo de ejecución: 0.31 s ± 0.021 s	

Tabla A.2: Resultados detallados (media ± desviación típica) de la validación cruzada para el modelo Item-KNN sobre ambos conjuntos de datos.

Métrica	**K=1**	**K=20**	**K=100**	**K=300**
Secuencias Originales				
HR@K (%)	0.16 ± 0.02	15.58 ± 0.95	55.17 ± 1.65	77.19 ± 1.35
MRR@K	0.0016 ± 0.0002	0.0217 ± 0.0018	0.0304 ± 0.0022	0.0318 ± 0.0022
NDCG@K	0.0016 ± 0.0002	0.0497 ± 0.0031	0.1202 ± 0.0045	0.1502 ± 0.0041
			Tiempo de ejecución: 3.43 s ± 0.15 s	
Secuencias Truncadas				
HR@K (%)	0.33 ± 0.04	12.16 ± 0.82	44.25 ± 1.24	59.91 ± 1.51
MRR@K	0.0033 ± 0.0004	0.0190 ± 0.0015	0.0258 ± 0.0019	0.0268 ± 0.0020
NDCG@K	0.0033 ± 0.0004	0.0404 ± 0.0028	0.0971 ± 0.0039	0.1189 ± 0.0042
			Tiempo de ejecución: 1.75 s ± 0.09 s	

Tabla A.3: Resultados detallados (media ± desviación típica) de la validación cruzada para el modelo User-KNN sobre ambos conjuntos de datos.

Métrica	K=1	K=20	K=100	K=300
Secuencias Originales				
HR@K (%)	0.00 ± 0.00	11.37 ± 0.78	51.99 ± 1.35	75.68 ± 1.12
MRR@K	0.0000 ± 0.0000	0.0121 ± 0.0011	0.0214 ± 0.0016	0.0229 ± 0.0016
NDCG@K	0.0000 ± 0.0000	0.0328 ± 0.0024	0.1062 ± 0.0038	0.1383 ± 0.0035
				Tiempo de ejecución: 3.26 s ± 0.12 s
Secuencias Truncadas				
HR@K (%)	0.22 ± 0.03	12.27 ± 0.85	50.16 ± 1.29	72.40 ± 1.41
MRR@K	0.0022 ± 0.0003	0.0165 ± 0.0013	0.0247 ± 0.0018	0.0261 ± 0.0019
NDCG@K	0.0022 ± 0.0003	0.0385 ± 0.0029	0.1059 ± 0.0041	0.1359 ± 0.0040
				Tiempo de ejecución: 2.66 s ± 0.10 s

Tabla A.4: Resultados detallados (media ± desviación típica) de la validación cruzada para el modelo SVD sobre ambos conjuntos de datos.

A.1.2. Redes secuenciales

En este apartado se presentan los resultados detallados correspondientes a las arquitecturas secuenciales evalueadas. Específicamente, se introducen las métricas obtenidas para el modelo Transformer (Tabla A.5), así como para Mamba (Tabla A.6), y xLSTM (Tabla A.7).

Métrica	K=1	K=20	K=100	K=300
Secuencias Originales				
HR@K (%)	8.16 ± 0.90	32.58 ± 1.27	56.55 ± 1.35	74.78 ± 1.56
MRR@K	0.0816 ± 0.0090	0.1326 ± 0.0101	0.1382 ± 0.0101	0.1393 ± 0.0101
NDCG@K	0.0816 ± 0.0090	0.1753 ± 0.0105	0.2186 ± 0.0106	0.2433 ± 0.0106
				Tiempo de entrenamiento: 8.60 min ± 22.56 s
Secuencias Truncadas				
HR@K (%)	5.30 ± 0.58	24.04 ± 1.06	48.37 ± 1.70	69.78 ± 1.89
MRR@K	0.0530 ± 0.0058	0.0902 ± 0.0055	0.0958 ± 0.0057	0.0971 ± 0.0056
NDCG@K	0.0530 ± 0.0058	0.1231 ± 0.0063	0.1670 ± 0.0073	0.1959 ± 0.0067
				Tiempo de entrenamiento: 6.10 min ± 12.16 s

Tabla A.5: Resultados detallados (media ± desviación típica) de la validación cruzada para el modelo Transformer.

Métrica	K=1	K=20	K=100	K=300
Secuencias Originales				
HR@K (%)	6.70 ± 0.65	29.04 ± 1.47	51.29 ± 1.83	69.20 ± 1.50
MRR@K	0.0670 ± 0.0065	0.1136 ± 0.0091	0.1187 ± 0.0091	0.1198 ± 0.0091
NDCG@K	0.0670 ± 0.0065	0.1526 ± 0.0100	0.1929 ± 0.0101	0.2170 ± 0.0097
				Tiempo de entrenamiento: 3.20 min ± 1.14 s
Secuencias Truncadas				
HR@K (%)	4.11 ± 0.31	20.26 ± 0.99	42.43 ± 1.64	62.90 ± 1.36
MRR@K	0.0411 ± 0.0031	0.0718 ± 0.0037	0.0767 ± 0.0037	0.0780 ± 0.0036
NDCG@K	0.0411 ± 0.0031	0.1003 ± 0.0042	0.1399 ± 0.0050	0.1675 ± 0.0045
				Tiempo de entrenamiento: 2.60 min ± 7.79 s

Tabla A.6: Resultados detallados (media ± desviación típica) de la validación cruzada para el modelo Mamba.

Métrica	K=1	K=20	K=100	K=300
Secuencias Originales				
HR@K (%)	7.83 ± 1.10	31.09 ± 1.31	54.28 ± 1.53	72.25 ± 1.24
MRR@K	0.0783 ± 0.0110	0.1277 ± 0.0109	0.1331 ± 0.0109	0.1342 ± 0.0109
NDCG@K	0.0783 ± 0.0110	0.1682 ± 0.0108	0.2102 ± 0.0114	0.2344 ± 0.0108
				Tiempo de entrenamiento: 7.30 min ± 9.63 s
Secuencias Truncadas				
HR@K (%)	4.83 ± 0.49	23.63 ± 1.22	47.30 ± 1.69	67.67 ± 1.66
MRR@K	0.0483 ± 0.0049	0.0860 ± 0.0037	0.0914 ± 0.0038	0.0926 ± 0.0038
NDCG@K	0.0483 ± 0.0049	0.1189 ± 0.0047	0.1615 ± 0.0051	0.1890 ± 0.0048
				Tiempo de entrenamiento: 5.70 min ± 9.29 s

Tabla A.7: Resultados detallados (media ± desviación típica) de la validación cruzada para el modelo xLSTM.

A.1.3. Redes neuronales de grafos

En este bloque se exponen las métricas respectivas a los modelos basados en grafos. En concreto, se recogen los resultados para la red convolucional de grafos clásica GCN (Tabla A.8), su variante optimizada LightGCN (Tabla A.9), y el modelo equipado con mecanismos de atención dinámicos GAT (Tabla A.10).

Métrica	K=1	K=20	K=100	K=300
Secuencias Originales				
HR@K (%)	1.40 ± 0.36	13.17 ± 1.05	31.56 ± 1.11	52.19 ± 1.58
MRR@K	0.0140 ± 0.0036	0.0342 ± 0.0041	0.0382 ± 0.0040	0.0394 ± 0.0040
NDCG@K	0.0140 ± 0.0036	0.0552 ± 0.0049	0.0878 ± 0.0048	0.1155 ± 0.0056
				Tiempo de entrenamiento: 4.97 s ± 0.17 s
Secuencias Truncadas				
HR@K (%)	0.80 ± 0.25	8.26 ± 1.01	26.92 ± 1.60	47.31 ± 2.00
MRR@K	0.0080 ± 0.0025	0.0207 ± 0.0030	0.0248 ± 0.0029	0.0260 ± 0.0029
NDCG@K	0.0080 ± 0.0025	0.0340 ± 0.0041	0.0672 ± 0.0042	0.0945 ± 0.0037
				Tiempo de entrenamiento: 3.46 s ± 0.24 s

Tabla A.8: Resultados detallados (media ± desviación típica) de la validación cruzada para la arquitectura de grafos GCN.

Métrica	K=1	K=20	K=100	K=300
Secuencias Originales				
HR@K (%)	1.85 ± 0.39	14.46 ± 1.16	34.17 ± 1.57	54.24 ± 1.76
MRR@K	0.0185 ± 0.0039	0.0410 ± 0.0048	0.0455 ± 0.0050	0.0467 ± 0.0050
NDCG@K	0.0185 ± 0.0039	0.0633 ± 0.0058	0.0988 ± 0.0068	0.1258 ± 0.0070
				Tiempo de entrenamiento: 4.78 s ± 0.19 s
Secuencias Truncadas				
HR@K (%)	0.83 ± 0.23	9.55 ± 0.60	27.67 ± 1.56	46.46 ± 2.21
MRR@K	0.0083 ± 0.0023	0.0216 ± 0.0032	0.0256 ± 0.0032	0.0268 ± 0.0032
NDCG@K	0.0083 ± 0.0023	0.0373 ± 0.0035	0.0697 ± 0.0042	0.0949 ± 0.0041
				Tiempo de entrenamiento: 3.39 s ± 0.09 s

Tabla A.9: Resultados detallados (media ± desviación típica) de la validación cruzada para la arquitectura de grafos LightGCN.

Métrica	K=1	K=20	K=100	K=300
Secuencias Originales				
HR@K (%)	1.89 ± 0.51	18.12 ± 3.08	46.00 ± 5.41	67.78 ± 4.75
MRR@K	0.0189 ± 0.0051	0.0452 ± 0.0095	0.0516 ± 0.0101	0.0529 ± 0.0101
NDCG@K	0.0189 ± 0.0051	0.0744 ± 0.0140	0.1245 ± 0.0184	0.1539 ± 0.0176
				Tiempo de entrenamiento: 5.50 min ± 127.60 s
Secuencias Truncadas				
HR@K (%)	1.07 ± 0.32	14.57 ± 1.19	41.28 ± 2.58	65.13 ± 3.04
MRR@K	0.0107 ± 0.0032	0.0312 ± 0.0051	0.0371 ± 0.0053	0.0386 ± 0.0053
NDCG@K	0.0107 ± 0.0032	0.0556 ± 0.0065	0.1033 ± 0.0085	0.1354 ± 0.0087
				Tiempo de entrenamiento: 4.40 min ± 79.40 s

Tabla A.10: Resultados detallados (media ± desviación típica) de la validación cruzada para la arquitectura de grafos GAT.

A.1.4. Modelos híbridos

Finalmente, en este apartado se recopilan los resultados asociados a los modelos híbridos que combinan los modelos secuenciales con información contextual externa. De este modo, se muestran las métricas para el modelo con apoyo semántico Transformer+BERT (Tabla A.11), y las arquitecturas con apoyo colaborativo mediante Transformer+GAT (Tabla A.12) y xLSTM+GAT (Tabla A.13).

Métrica	K=1	K=20	K=100	K=300
Secuencias Originales				
HR@K (%)	7.66 ± 0.76	34.33 ± 1.53	59.14 ± 1.24	76.23 ± 1.45
MRR@K	0.0766 ± 0.0076	0.1321 ± 0.0093	0.1380 ± 0.0091	0.1391 ± 0.0091
NDCG@K	0.0766 ± 0.0076	0.1788 ± 0.0103	0.2239 ± 0.0091	0.2470 ± 0.0095
				Tiempo de entrenamiento: 7.33 min ± 14.41 s
Secuencias Truncadas				
HR@K (%)	4.91 ± 0.34	25.92 ± 1.44	51.27 ± 1.37	72.08 ± 1.14
MRR@K	0.0491 ± 0.0034	0.0904 ± 0.0044	0.0962 ± 0.0042	0.0975 ± 0.0042
NDCG@K	0.0491 ± 0.0034	0.1273 ± 0.0065	0.1729 ± 0.0055	0.2010 ± 0.0050
				Tiempo de entrenamiento: 5.13 min ± 15.20 s

Tabla A.11: Resultados detallados (media ± desviación típica) de la validación cruzada para el modelo híbrido Transformer+BERT.

Métrica	K=1	K=20	K=100	K=300
Secuencias Originales				
HR@K (%)	8.54 ± 0.82	34.84 ± 1.54	59.75 ± 1.93	77.38 ± 1.69
MRR@K	0.0854 ± 0.0082	0.1412 ± 0.0098	0.1471 ± 0.0096	0.1482 ± 0.0096
NDCG@K	0.0854 ± 0.0082	0.1870 ± 0.0105	0.2323 ± 0.0102	0.2561 ± 0.0102
				Tiempo de entrenamiento: 9.30 min ± 22.73 s
Secuencias Truncadas				
HR@K (%)	5.69 ± 0.69	25.76 ± 1.26	52.07 ± 1.12	72.48 ± 1.42
MRR@K	0.0569 ± 0.0069	0.0966 ± 0.0066	0.1025 ± 0.0063	0.1038 ± 0.0063
NDCG@K	0.0569 ± 0.0069	0.1319 ± 0.0074	0.1790 ± 0.0060	0.2066 ± 0.0055
				Tiempo de entrenamiento: 6.30 min ± 18.78 s

Tabla A.12: Resultados detallados (media ± desviación típica) de la validación cruzada para el modelo híbrido Transformer+GAT.

Métrica	K=1	K=20	K=100	K=300
Secuencias Originales				
HR@K (%)	8.98 ± 1.04	36.11 ± 1.76	59.63 ± 1.56	76.03 ± 1.43
MRR@K	0.0898 ± 0.0104	0.1471 ± 0.0120	0.1527 ± 0.0118	0.1537 ± 0.0118
NDCG@K	0.0898 ± 0.0104	0.1945 ± 0.0129	0.2373 ± 0.0116	0.2594 ± 0.0116
				Tiempo de entrenamiento: 15.40 min ± 41.14 s
Secuencias Truncadas				
HR@K (%)	6.17 ± 0.47	27.54 ± 1.34	52.19 ± 1.10	71.62 ± 1.84
MRR@K	0.0617 ± 0.0047	0.1042 ± 0.0044	0.1098 ± 0.0045	0.1110 ± 0.0044
NDCG@K	0.0617 ± 0.0047	0.1418 ± 0.0048	0.1861 ± 0.0046	0.2124 ± 0.0047
				Tiempo de entrenamiento: 12.10 min ± 31.20 s

Tabla A.13: Resultados detallados (media ± desviación típica) de la validación cruzada para el modelo híbrido xLSTM+GAT.

A.2. Gráficos de personalización

En esta última sección del apéndice se proporcionan las representaciones visuales referentes a la capacidad de personalización de las distintas arquitecturas evaluadas. Específicamente, se incluyen las estimaciones tipo núcleo de la densidad que comparan la distribución empírica de popularidad de las películas del conjunto de datos original con las recomendaciones generadas por el algoritmo Item-KNN (Figura A.1), los modelos secuenciales Mamba (Figura A.2) y xLSTM (Figura A.3), las arquitecturas basadas en grafos GCN (Figura A.4) y GAT (Figura A.5), y finalmente el modelo híbrido xLSTM+GAT (Figura A.6).

El área sombreada representa el comportamiento real de los usuarios, mientras que la línea discontinua ilustra las recomendaciones del modelo. La leyenda detalla la popularidad media (número medio de

apariciones en secuencias) de los ítems, en tanto por mil.

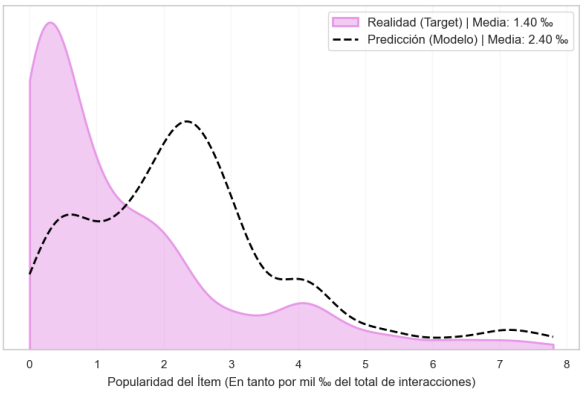

Figura A.1: Estimación tipo núcleo de la densidad de la distribución de popularidad para las películas en el conjunto de secuencias de MovieLens comparado con la predicción del modelo ItemKNN.

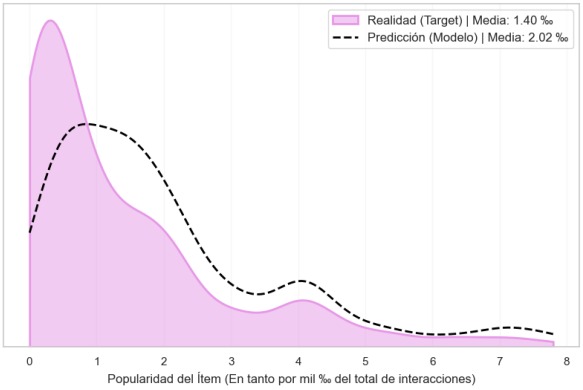

Figura A.2: Estimación tipo núcleo de la densidad de la distribución de popularidad para las películas en el conjunto de secuencias de MovieLens comparado con la predicción del modelo Mamba.

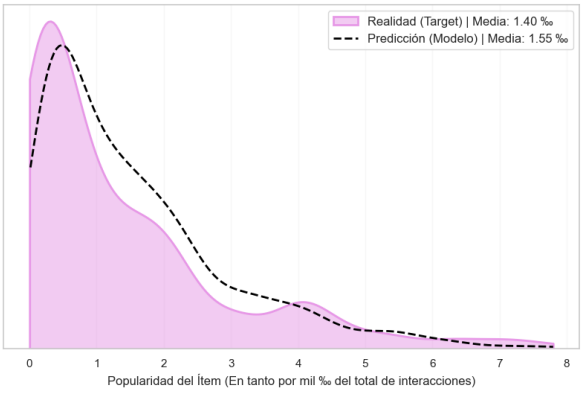

Figura A.3: Estimación tipo núcleo de la densidad de la distribución de popularidad para las películas en el conjunto de secuencias de MovieLens comparado con la predicción del modelo xLSTM.

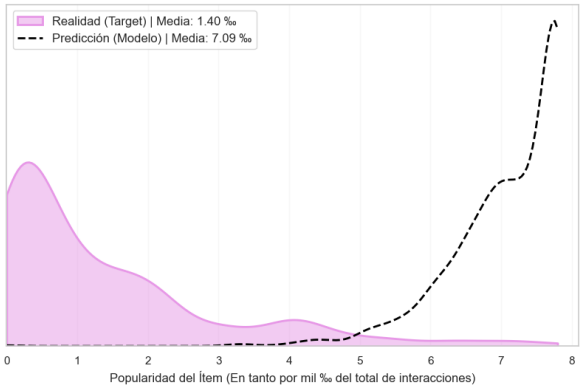

Figura A.4: Estimación tipo núcleo de la densidad de la distribución de popularidad para las películas en el conjunto de secuencias de MovieLens comparado con la predicción del modelo GCN.

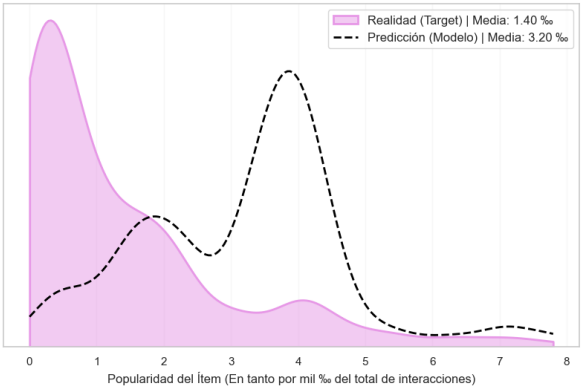

Figura A.5: Estimación tipo núcleo de la densidad de la distribución de popularidad para las películas en el conjunto de secuencias de MovieLens comparado con la predicción del modelo GAT.

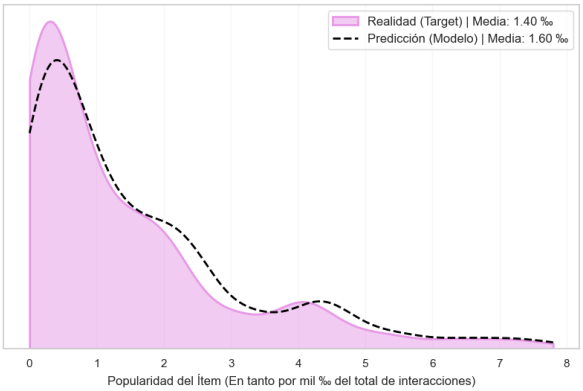

Figura A.6: Estimación tipo núcleo de la densidad de la distribución de popularidad para las películas en el conjunto de secuencias de MovieLens comparado con la predicción del modelo xLSTM+GAT.

Bibliografía

Bansal, Tanmay (2025). *Why Large Context AI Is So Expensive (And How Mamba Fixes It)*. URL: https://ai.plainenglish.io/why-million-token-ai-is-so-expensive-and-how-mamba-fixes-it-a69c849d580c (visitado 20-05-2026).

Beck, Maximilian, Korbinian Pöppel, Markus Spanring, Andreas Auer, Oleksandra Prudnikova, Michael Kopp, Günter Klambauer, Johannes Brandstetter y Sepp Hochreiter (2024). «xLSTM: Extended Long Short-Term Memory». *Advances in Neural Information Processing Systems*.

Cao, Longbing y Chengzhang Zhu (2022). «Personalized next-best action recommendation with multi-party interaction learning for automated decision-making». *PLOS ONE* 17.2, e0263010.

Covington, Paul, Jay Adams y Emre Sargin (2016). «Deep Neural Networks for YouTube Recommendations». *Proceedings of the 10th ACM Conference on Recommender Systems*. RecSys '16. Boston, Massachusetts, USA: Association for Computing Machinery, págs. 191-198.

Devlin, Jacob, Ming-Wei Chang, Kenton Lee y Kristina Toutanova (2019). «BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding». *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Ed. por Jill Burstein, Christy Doran y Thamar Solorio. Minneapolis, Minnesota: Association for Computational Linguistics, págs. 4171-4186.

Dosovitskiy, Alexey, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit y Neil Houlsby (2021). «An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale». *International Conference on Learning Representations*.

Fan, Yu-Chen, Yitong Ji, Jie Zhang y Aixin Sun (2024). «Our Model Achieves Excellent Performance on MovieLens: What Does It Mean?» *ACM Trans. Inf. Syst.* 42.6, págs. 1-25.

Goodfellow, Ian, Yoshua Bengio y Aaron Courville (2016). *Deep learning*. MIT press.

Grootendorst, Maarten (2024). *A Visual Guide to Mamba and State Space Models*. URL: https://maartengrootendorst.substack.com/p/a-visual-guide-to-mamba-and-state (visitado 01-02-2026).

Gu, Albert y Tri Dao (2024). «Mamba: Linear-Time Sequence Modeling with Selective State Spaces». *Proceedings of the 41st International Conference on Machine Learning (ICML)*. Vol. 235. Proceedings of Machine Learning Research. PMLR, págs. 15918-15943.

Hagberg, Aric A., Daniel A. Schult y Pieter J. Swart (2008). «Exploring Network Structure, Dynamics, and Function using NetworkX». *Proceedings of the 7th Python in Science Conference*. Pasadena, CA USA, págs. 11-15.

Harper, F. Maxwell y Joseph A. Konstan (2015). «The MovieLens Datasets: History and Context». *ACM Transactions on Interactive Intelligent Systems* 5.4, págs. 1-19.

He, Xiangnan, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang y Meng Wang (2020). «LightGCN: Simplifying and powering graph convolution network for recommendation». *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, págs. 639-648.

Hochreiter, Sepp y Jürgen Schmidhuber (1997). «Long short-term memory». *Neural computation* 9.8, págs. 1735-1780.

Hugging Face (2025). *Hugging Face Hub*. URL: https://huggingface.co/ (visitado 25-10-2025).

IMDb.com, Inc. (2026). *Internet Movie Database*. URL: https://www.imdb.com/ (visitado 07-11-2025).

Järvelin, Kalervo y Jaana Kekäläinen (2002). «Cumulated gain-based evaluation of IR techniques». *ACM Transactions on Information Systems* 20.4, págs. 422-446.

Katharopoulos, Angelos, Apoorv Vyas, Nikolaos Pappas y François Fleuret (2020). «Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention». *International Conference on Machine Learning*. PMLR, págs. 5281-5292.

Kipf, Thomas N. y Max Welling (2017). «Semi-Supervised Classification with Graph Convolutional Networks». *International Conference on Learning Representations*.

Letterboxd Limited (2011). *Letterboxd*. URL: https://letterboxd.com/ (visitado 03-03-2026).

Lieber, Opher, Barak Lenz, Harm De Vries, Ozan Mutlu, Itay Shaham, Chen Shacham, Guy Gur, Omer Alon, Mor Geva, Omer Levy y Yoav Shoham (2024). «Jamba: A Hybrid Transformer-Mamba Language Model». *Advances in Neural Information Processing Systems*. Vol. 37, págs. 89452-89478.

Lim, Bryan, Sercan O. Arik, Nicolas Loeff y Tomas Pfister (2021). «Temporal fusion transformers for interpretable multi-horizon time series forecasting». *International Journal of Forecasting* 37.4, págs. 1748-1764.

Loshchilov, Ilya y Frank Hutter (2019). «Decoupled Weight Decay Regularization». *International Conference on Learning Representations*.

Paszke, Adam, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai y Soumith Chintala (2019). «PyTorch: An Imperative Style, High-Performance Deep Learning Library». *Advances in Neural Information Processing Systems 32*, págs. 8024-8035.

Quadrana, Massimo, Paolo Cremonesi y Dietmar Jannach (2018). «Sequence-aware recommender systems». *ACM computing surveys* 51.4, págs. 1-36.

Radford, Alec, Jeff Wu, Rewon Child, David Luan, Dario Amodei e Ilya Sutskever (2019). *Language models are unsupervised multitask learners*. Inf. téc. OpenAI.

Rendle, Steffen, Christoph Freudenthaler, Zeno Gantner y Lars Schmidt-Thieme (2009). «BPR: Bayesian personalized ranking from implicit feedback». *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*, págs. 452-461.

Ricci, Francesco, Lior Rokach y Bracha Shapira, eds. (2015). *Recommender Systems Handbook*. Springer.

Rumelhart, David E, Geoffrey E Hinton y Ronald J Williams (1986). «Learning representations by back-propagating errors». *nature* 323.6088, págs. 533-536.

Salton, Gerard y Michael J McGill (1983). *Introduction to modern information retrieval*. McGraw-Hill.

Sarwar, Badrul, George Karypis, Joseph Konstan y John Riedl (2001). «Item-based collaborative filtering recommendation algorithms». *Proceedings of the 10th International Conference on World Wide Web*. ACM, págs. 285-295.

Streamlit Inc. (2026). *Streamlit: A faster way to build and share data apps*. URL: https://streamlit.io/ (visitado 28-02-2026).

Sutskever, Ilya, Oriol Vinyals y Quoc V Le (2014). «Sequence to Sequence Learning with Neural Networks». *Advances in Neural Information Processing Systems*. Vol. 27, págs. 3104-3112.

The Movie Database (2026). *TMDb*. URL: https://www.themoviedb.org/ (visitado 24-11-2025).

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser e Illia Polosukhin (2017). «Attention is all you need». *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., págs. 6000-6010.

Veličković, Petar, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò y Yoshua Bengio (2018). «Graph Attention Networks». *International Conference on Learning Representations*.

Wang, Wenhui, Furu Wei, Li Dong, Hangbo Bao, Minlie Yang y Ming Zhou (2020). «MiniLM: Deep Self-Attention Distillation for Task-Agnostic Compression of Pre-Trained Transformers». *Advances in Neural Information Processing Systems*. Vol. 33, págs. 5776-5788.

Warner, Benjamin, Antoine Chaffin, Benjamin Clavié, Orion Weller, Oskar Hallström, Said Taghadouini, Alexis Gallagher, Raja Biswas, Faisal Ladhak, Tom Aarsen, Nathan Cooper, Griffin Adams, Jeremy

Howard y Iacopo Poli (2025). «Smarter, Better, Faster, Longer: A Modern Bidirectional Encoder for Fast, Memory Efficient, and Long Context Finetuning and Inference». *International Conference on Learning Representations*.

Wolf, Thomas, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Cistac Pierric, Canwen Li, Raymond Tan, Huseyin Yilmaz, Tan Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Can Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest y Alexander M. Rush (2020). «Transformers: State-of-the-Art Natural Language Processing». *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, págs. 38-45.