# Decision Transformers para recomendaciones

Aprendizaje por Refuerzos

Felipe Andrés Ávila - Sofía Perón Santana

## 1 Introducción

El objetivo de este trabajo es realizar una implementación de *Decision Transformers* (DTs) para construir un sistema de recomendación similar a los utilizados por plataformas de *streaming* de contenido. Los modelos tipo *Transformer* destacan por su capacidad para representar relaciones complejas dentro de secuencias, por eso han sido especialmente exitosos en procesamiento de lenguaje natural. En 2021 se propuso extender esta idea a escenarios típicos de aprendizaje por refuerzo (entornos de OpenAI Gym, Atari, etc) [1], interpretando las trayectorias de interacción agente-entorno como secuencias a modelar. Se toma como referencia el trabajo de Rajapakse & Leith [2], que introduce RLT4Rec, un enfoque que toma como entrada ítems consumidos por usuarios junto con sus calificaciones, y utiliza un modelo *Transformer* para predecir cuál debería ser el siguiente ítem a presentar. La implementación completa puede encontrarse en [3].

### 1.1 Formulación de un sistema de recomendaciones como MDP y DT

Para trabajar un sistema de recomendaciones en el contexto de aprendizaje por refuerzos es necesario representarlo como un Proceso de Decisión de Markov (MDP). En general, estos procesos se modelan con una tupla $(\mathcal{S}, \mathcal{A}, P, r)$, cuyos elementos son, respectivamente, el conjunto de estados, el conjunto de acciones, la dinámica del entorno y la función de recompensa.

En este caso particular, se define el estado $s_t$ a partir de la información disponible en cada paso temporal: los últimos $k$ ítems o películas con los que el usuario interactuó, el grupo o *cluster* al que pertenece (ocho grupos de preferencias obtenidos mediante un método similar a *k-means*) y el *timestep* dentro de la sesión, que introduce contexto temporal. La acción $a_t$ corresponde a seleccionar qué película recomendar en el siguiente paso, lo que da lugar a un espacio de 752 acciones posibles (una por cada película presente en el conjunto de datos). La recompensa $r_t$ está dada por el *rating* del usuario, que puede tomar valores enteros entre 1 (malo) y 5 (excelente). Al trabajar en un escenario *offline*, en lugar de tener una dinámica del entorno que permita generar nuevos estados, se dispone únicamente de un conjunto fijo de trayectorias históricas y, por lo tanto, no existe la posibilidad de interacción ni exploración durante el entrenamiento.

En un DT, en lugar de aprender explícitamente una política que determine la acción óptima en cada estado, se entrena un modelo autoregresivo a partir de las trayectorias observadas. Las secuencias $(s_t, a_t)$, junto con el *return-to-go* definido como la suma de recompensas futuras, constituyen la entrada del modelo que se entrena para imitar políticas de alto rendimiento condicionándose en retornos deseados.

# 2 Dataset y Preprocesamiento

## 2.1 Descripción del Dataset

El conjunto de datos utilizado corresponde a un `DataFrame` de `pandas`. El archivo de entrenamiento, contiene 16000 usuarios identificados por un número entero (id), y 752 películas. Cada usuario pertenece a uno de los 8 grupos o *clusters*, definidos a partir de un algoritmo de *clustering* sobre sus preferencias de *rating*. Para cada usuario se observa una secuencia histórica de interacciones, constituyendo un total de aproximadamente 1797612 *ratings* en total.

La Figura 1 muestra el histograma de la longitud de las secuencias de interacción por usuario, es decir, la cantidad de películas vistas o calificadas por cada uno. La distribución es relativamente uniforme, lo que indica que prácticamente todos los usuarios presentan cadenas de interacción de longitud similar.

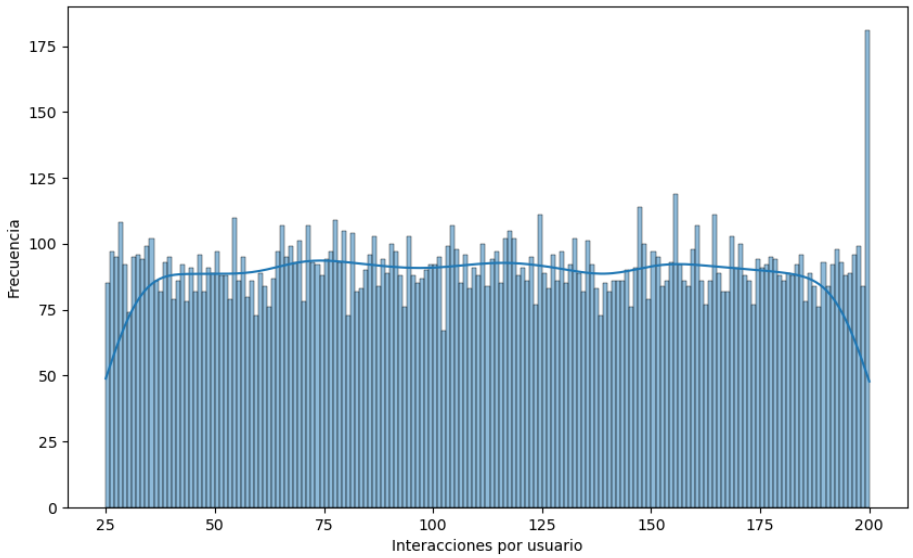


Figura 1: Distribución de la longitud de las secuencias de interacción por usuario.

La popularidad de las películas se analiza en la Figura 2. En el primer gráfico se representa la distribución del número de interacciones por película, mientras que en el segundo se muestra un diagrama de dispersión entre cantidad de interacciones por película y la calificación promedio de cada ítem. En el histograma se observa que la cantidad de interacciones uniforme: la mayoría de los ítems son calificados un número de veces similar. El diagrama de dispersión confirma que la relación entre popularidad y *rating* es débil: la correlación lineal entre frecuencia y calificación promedio es de $-0{,}032$), por lo que las películas más vistas no tienden sistemáticamente a ser mejor ni peor valoradas que las menos vistas.

Las 20 películas más populares presentan una calificación media de 3.05, mientras que las 20 menos populares alcanzan un valor medio de 3.28, coherente con la baja correlación observada en la Figura 2. La Tabla 1 resume los 10 ítems con más y menos interacciones, indicando su frecuencia y su *rating* promedio. Los *rankings* completos de todo el *dataset* junto con los correspondientes a cada *cluster* pueden encontrarse aquí.

La distribución de *ratings* se analiza tanto a nivel global como por grupo de usuarios en la Figura 3. Considerando todos los usuarios, las calificaciones se reparten de forma relativamente homogénea entre los valores 1 y 5, aunque con una ligera tendencia hacia *ratings* altos (4 y 5).

Al desagregar por *cluster* aparecen patrones de comportamiento diferenciados. Algunos (0, 1, 4 y

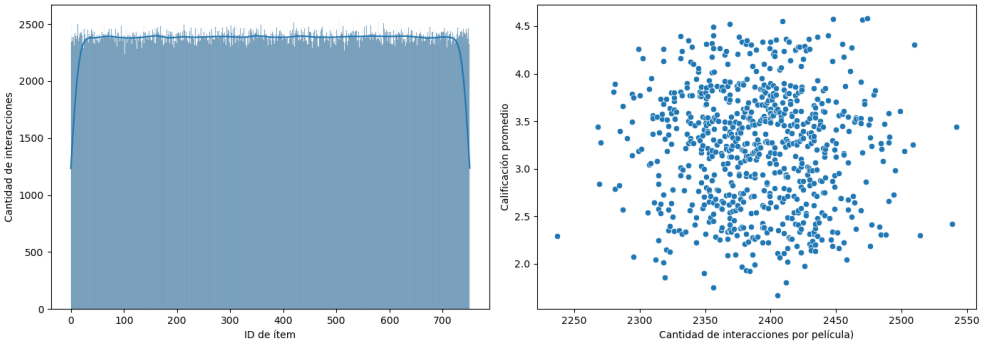


Figura 2: Popularidad de ítems y relación entre frecuencia de interacción y calificación promedio por película.

Tabla 1: Películas con más y menos interacciones: frecuencia y calificación promedio.

| Películas con más interacciones | | | | | | Películas con menos interacciones | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| *item* | *frec* | *rating* | *item* | *frec* | *rating* | *item* | *frec* | *rating* | *item* | *frec* | *rating* |
| 505 | 2542 | 3.44 | 579 | 2539 | 2.42 | 101 | 2237 | 2.29 | 317 | 2268 | 3.44 |
| 420 | 2514 | 2.30 | 532 | 2510 | 4.30 | 462 | 2269 | 2.84 | 515 | 2270 | 3.27 |
| 589 | 2509 | 3.25 | 63 | 2502 | 3.18 | 124 | 2280 | 3.81 | 540 | 2281 | 2.79 |
| 13 | 2499 | 3.61 | 622 | 2495 | 2.98 | 110 | 2281 | 3.89 | 455 | 2284 | 2.82 |
| 358 | 2494 | 2.73 | 213 | 2491 | 3.34 | 23 | 2285 | 3.40 | 134 | 2287 | 3.66 |

6), muestran una clara inclinación a otorgar calificaciones altas. Otros grupos (2, el 3 y el 5), presentan distribuciones más uniformes, sin un sesgo tan marcado. Finalmente, el grupo 7 se caracteriza por una concentración importante de *ratings* bajos (1 y 2). Estas diferencias justifican la inclusión del `user_group` como variable de contexto en el modelo, ya que capturan preferencias y criterios de evaluación heterogéneos entre subconjuntos de usuarios.

### 2.2 Preprocesamiento

Siguiendo la formulación del sistema de recomendación como MDP y *Decision Transformer* presentada en la Sección 1.1, se aplicó la función `create_dt_dataset`, sobre el `DataFrame` original. Esta función recorre cada usuario y construye una trayectoria en el formato requerido por el modelo, que incluye la secuencia completa de ítems y sus *ratings*, el vector de *return-to-go* (suma acumulada hacia adelante), los índices temporales (*timesteps*) y el *cluster* correspondiente. Para mejorar el desempeño del modelo, se normalizaron las recompensas acumuladas *return-to-go* (rtg) respecto de la media según $(\hat{R} - \bar{\hat{R}})/(\sigma_{\hat{R}} + \delta)$ donde  es el promedio sobre todas las rtg con desviación estándar $(\sigma_{\hat{R}})$ y el factor $\delta$ se agrega para estabilidad numérica.

## 3 Implementación

A continuación se describe el entrenamiento del *Decision Transformer* utilizado. El procedimiento general es el siguiente: el `DataLoader` divide el conjunto de datos en secuencias de longitud `context_length` y las organiza en *batches* para el entrenamiento. Para cada *batch* se toman como entrada los estados, acciones, RTGs, *timesteps* y grupos, a los que se les aplican los correspondientes *embeddings*. Luego se construye la secuencia intercalada y se suman los *embeddings* de cada *token*. Tras normalizar el resultado, la secuencia

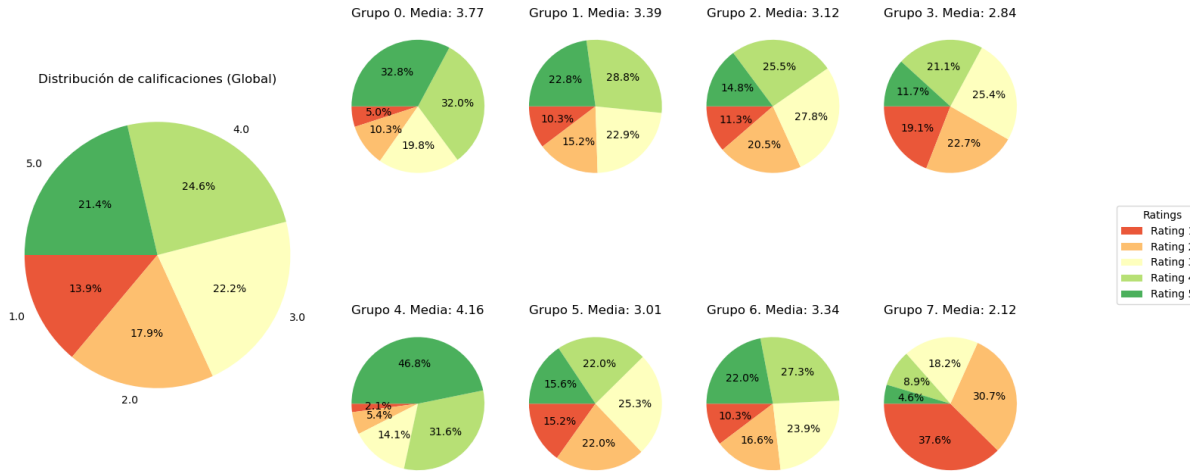


Figura 3: Distribución de ratings global y por grupo de usuarios.

se procesa con el *transformer* empleando una máscara causal, y a partir de los estados se predice la siguiente recomendación, utilizando *cross-entropy* como función de pérdida.

### 3.1 Arquitectura del Decision Transformer

La clase `DecisionTransformer`, implementada en el módulo `src/models/decision_transformer.py`, define los *embeddings* asociados a los distintos tipos de dato. Se utiliza una tabla de *embeddings* para los estados, otra para las acciones, y una transformación lineal para mapear el valor escalar del *return-to-go* a un espacio vectorial. Además de los *embeddings* de contenido, se incorporan tres tipos adicionales de información contextual: un *token type embedding* que indica si el *token* corresponde a un RTG, a un estado o a una acción; un *timestep embedding*, que codifica la posición temporal y un *group embedding* para el *cluster* del usuario.

```
class DecisionTransformer(nn.Module):
    def __init__(...):
        # Embeddings independientes para cada tipo de token
        self.state_embedding = nn.Embedding(num_items, hidden_dim)
        self.action_embedding = nn.Embedding(num_items, hidden_dim)
        self.rtg_embedding = nn.Linear(1, hidden_dim)
        # Embeddings de contexto
        self.group_embedding = nn.Embedding(num_groups, hidden_dim)
        self.timestep_embedding = nn.Embedding(max_timestep, hidden_dim)
        # Token type embeddings (RTG, STATE, ACTION)
        self.token_type_embeddings = nn.Embedding(3, hidden_dim)
```

Posteriormente, en el método `forward` los *embeddings* de estado, acción y RTG de cada paso temporal se intercalan (*interleaving*). Así, dada una ventana de contexto de longitud $L$ la secuencia efectiva que ingresa al modelo posee longitud $3L$.

```
    def forward(...):
        # INTERLEAVING: [RTG_0, STATE_0, ACTION_0, RTG_1, STATE_1, ACTION_1, ...]
        tokens_list = []
        for t in range(seq_len):
```

```
            tokens_list.extend([rtg_embedding[:, t:t+1, :],     # (B, 1, H)
                                state_embedding[:, t:t+1, :],   # (B, 1, H)
                                action_embedding[:, t:t+1, :]]) # (B, 1, H)
        h = torch.cat(tokens_list, dim=1)  # (B, L*3, H) # concatenar
        (...) # se extienden los embeddings de interleaving y token_type
        h = h + group_emb +  timestep_emb_interleaved + token_type_emb
```

Finalmente, se emplea una máscara causal de modo que cada *token* solo puede "ver" *tokens* correspondientes a pasos temporales anteriores. Dentro del mismo bloque temporal se respeta un orden específico donde `RTG` solo ve `RTG`s previos, `STATE` puede atender a `RTG` y `STATE` del mismo instante, y `ACTION` puede atender a todos los *tokens* de su propio paso temporal. Se incluye la estructura *interleaved* explícitamente.

El modelo utiliza un *transformer* encoder de 2 capas, donde cada capa aplica atención multi-cabezal y una red *feed-forward*. Estas capas permiten que el modelo relacione distintas partes de la secuencia y aprenda qué información pasada es relevante para predecir la siguiente recomendación.

### 3.2 Modificaciones realizadas al código de referencia

Se compara la implementación con la que fue provista en la consigna. Para alcanzar un entrenamiento estable y permitir que el modelo redujera efectivamente la función de pérdida, fue necesario modificar diversos aspectos de la arquitectura original. Las diferencias más importantes entre ambas implementaciones fueron los siguientes:

- ***Embeddings* separados por tipo**: se reemplazó el uso de un único *item embedding* para estados y acciones por *embeddings* independientes para `state`, `action` y `RTG`, evitando la superposición semántica entre tokens.
- **Estructura intercalada de *tokens***: la codificación original combinaba los *embeddings* por suma; la versión final construye explícitamente la secuencia intercalada `[RTG_t, state_t, action_t]`, replicando la formulación del *Decision Transformer* original [1].
- ***Token type embeddings***: se incorporaron *embeddings* de tipo de *token* (RTG, state, action) que permiten al modelo distinguir claramente el rol funcional de cada entrada.
- **Inicialización de pesos**: se añadió una inicialización más controlada.
- **Simplificación de la capa de predicción**: se reemplazó la capa de predicción profunda por una proyección lineal directa, reduciendo la complejidad buscando evitar sobreajuste temprano.
- **Ajustes en el *dataset***: se reorganizó la preparación de estados, acciones, retornos (normalizando estos) y *targets*, asegurando consistencia con la estructura intercalada del modelo y evitando desalineamientos temporales.
- **Nuevos híper-parámetros**: A pesar de no haber implementado un barrido grillado, se realizaron pruebas sobre distintos conjuntos.

### 3.3 Hiper-parámetros utilizados

La siguiente tabla resume los híper-parámetros utilizados para el entrenamiento reportado junto con los del modelo de referencia. Se realizaron distintas pruebas para ambos, pero solo se reportan los que proveyeron mejores resultados.

| Hiperparámetro | Descripción | Valor | Referencia |
|---|---|---|---|
| `num_items` | Cantidad total de ítems (entradas). | 752 | 752 |
| `num_groups` | Cantidad de grupos o segmentos de usuario. | 8 | 8 |
| `hidden_dim` | Dimensión del espacio latente. | 512 | 128 |
| `n_layers` | Cuántas veces se repite la arquitectura interna. | 2 | 3 |
| `n_heads` | Número de cabezas de atención multi-head. | 4 | 4 |
| `context_length` | Longitud de la ventana de contexto. | 25 | 20 |
| `max_timestep` | Máximo *timestep* para *embeddings* temporales. | 200 | 200 |
| `dropout` | Probabilidad de *dropout* en el *Transformer*. | 0.05 | 0.1 |
| `batch_size` | Tamaño de *batch* durante el entrenamiento. | 64 | 32 |
| `learning_rate` | Tasa de aprendizaje del optimizador. | 1E-4 | 1E-5 a 1E-4 |
| `gradient_clip` | Límite de norma para *clipping* de gradientes. | 0.5 | 1 |

Tabla 2: Híper-parámetros utilizados para el entrenamiento

# 4 Resultados y Discusión

## 4.1 Resultados de entrenamiento

El modelo *Decision Transformer* se entrenó durante 1000 y 2000 épocas para el código refactorizado y el de referencia, respectivamente. La función de pérdida empleada fue la *cross-entropy* entre la distribución de probabilidad sobre los 752 ítems producida por el modelo y la acción objetivo (el ítem efectivamente observado en el siguiente paso), promediada sobre todos los pasos temporales y ejemplos del *batch*. Esta función de *loss* mide qué tan bien el modelo asigna alta probabilidad al ítem correcto en cada paso de la secuencia. Para la versión modificada, se usa Adamw (con w = 0.01) en lugar de Adam.

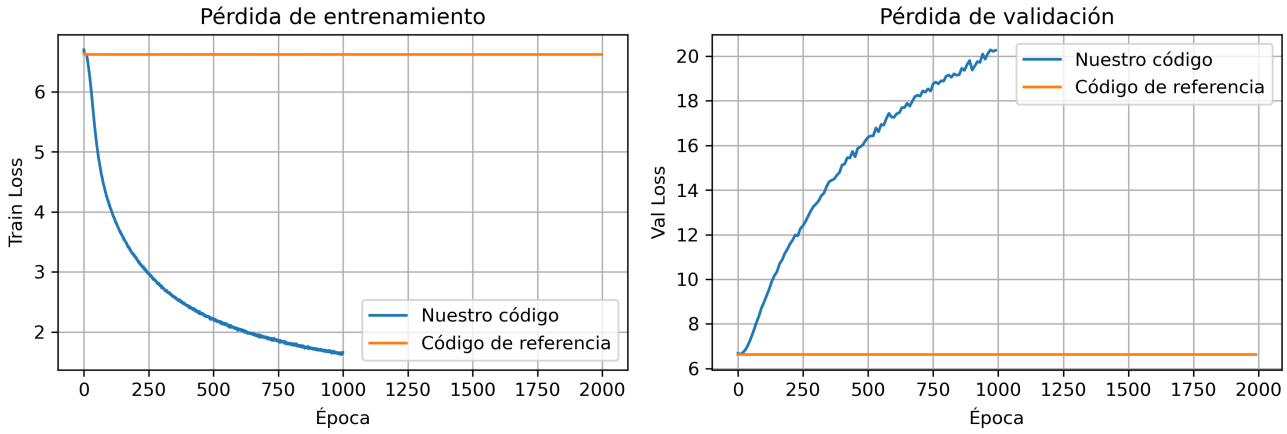


Figura 4: Comparación de funciones de perdida de entrenamiento y validación.

Como se observa en la figura 4, se observa un fuerte decrecimiento en la pérdida de entrenamiento que no se refleja en la validación, indicando que el modelo está sobreajustando.

## 4.2 Evaluación cuantitativa y comparación con el *baseline* de popularidad

Para evaluar la calidad de las recomendaciones se comparó el *Decision Transformer* con un modelo de popularidad que recomienda, para cada usuario, los ítems más frecuentes en el conjunto de entrenamiento. Ambos modelos se evaluaron en el *test set* de usuarios *cold-start*. La Tabla 3 muestra los resultados

obtenidos para las métricas HR@5, HR@10, HR@20, NDCG@5, NDCG@10, NDCG@20 y MRR.

Tabla 3: Resultados de evaluación en el conjunto de prueba (usuarios *cold-start*).

| **Métrica** | **Modelo final** | **Modelo de referencia** | **Baseline** |
|---|---|---|---|
| HR@5 | 0.008583 | 0.006831 | 0.006878 |
| NDCG@5 | 0.005364 | 0.004028 | 0.004116 |
| HR@10 | 0.015107 | 0.013661 | 0.012380 |
| NDCG@10 | 0.007484 | 0.006207 | 0.005852 |
| HR@20 | 0.028461 | 0.027322 | 0.026135 |
| NDCG@20 | 0.010818 | 0.009618 | 0.009245 |
| MRR | 0.010695 | 0.009726 | 0.009730 |

Las métricas HR@K (*Hit Rate*) miden la proporción de casos en los que el ítem correcto aparece dentro de las $K$ recomendaciones principales del modelo. Por ejemplo, HR@10 indica en qué fracción de las interacciones el ítem que el usuario realmente seleccionó se encuentra dentro del top-10 de ítems recomendados. Es una métrica binaria por interacción (acierto o no acierto) y toma valores entre 0 y 1.

Las métricas NDCG@K (*Normalized Discounted Cumulative Gain*) también consideran sólo las $K$ mejores recomendaciones, pero además tienen en cuenta la posición del ítem correcto dentro del *ranking*: se otorga mayor peso si aparece en las primeras posiciones y se penaliza si sólo aparece en posiciones bajas. El valor se normaliza para quedar en el rango $[0, 1]$, donde 1 corresponde al *ranking* ideal en el que el ítem correcto ocupa la primera posición.

Finalmente, la métrica MRR (*Mean Reciprocal Rank*) evalúa la calidad del *ranking* completo de ítems. Para cada interacción se calcula el inverso de la posición del ítem correcto (1/rank) y luego se promedia. Un valor cercano a 1 indica que el ítem correcto suele aparecer muy arriba en la lista, mientras que valores cercanos a 0 implican posiciones consistentemente bajas.

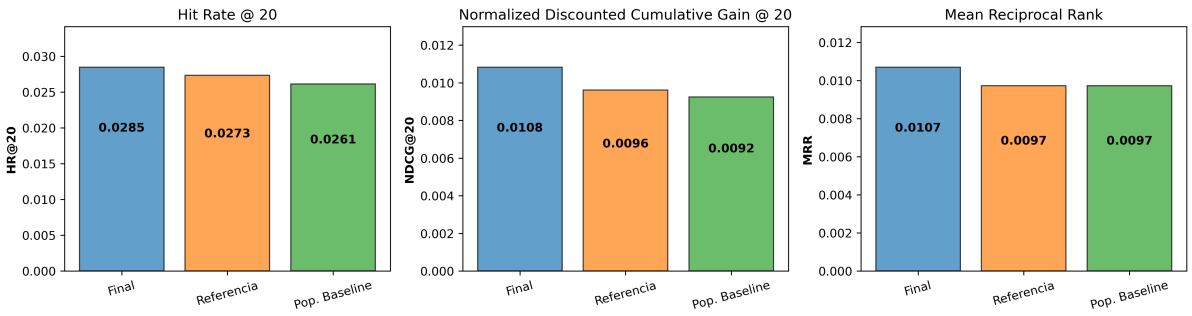


Figura 5: Comparación para rangos amplios entre el *Decision Transformer*, el modelo de referencia y el *baseline* de popularidad.

En nuestros experimentos, los valores absolutos de las métricas son bajos, lo que refleja la dificultad del escenario *cold-start* con 752 ítems posibles. Sin embargo, se observa que el *Decision Transformer* supera sistemáticamente al *baseline* de popularidad en todas las métricas evaluadas. Por ejemplo, HR@10 mejora de 0.0124 (popularidad) a 0.0151, mientras que NDCG@10 pasa de 0.00585 a 0.00748. El mismo patrón se repite para HR@5, HR@20, NDCG@5, NDCG@20 y MRR. Esto indica que, incluso en un entorno nuevo, el modelo es capaz de aprovechar información secuencial para producir recomendaciones más útiles que un método no personalizado basado únicamente en frecuencias globales.

### 4.3 Análisis del condicionamiento por retorno

Para estudiar el efecto del condicionamiento por retorno, se evaluó el *Decision Transformer* fijando distintos valores de `target_return` correspondientes a percentiles de la distribución de retornos observados. La Tabla 4 resume los resultados obtenidos sobre el conjunto de prueba.

Tabla 4: Rendimiento del *Decision Transformer* para distintos valores de `target_return`.

| **Percentil** | $R$ | **HR@10** | **NDCG@10** | **MRR** |
|---|---|---|---|---|
| $p_{25}$ | 213 | 0.0118 | 0.0050 | 0.0086 |
| $p_{50}$ | 350 | 0.0120 | 0.0050 | 0.0086 |
| $p_{75}$ | 497 | 0.0102 | 0.0049 | 0.0100 |
| $p_{90}$ | 604 | 0.0108 | 0.0049 | 0.0097 |
| max | 849 | 0.0094 | 0.0042 | 0.0094 |

Los retornos considerados cubren un rango amplio (entre 213 y 849), lo cual permite evaluar cómo responde el modelo a diferentes niveles de `target_return`. Sin embargo, las diferencias entre configuraciones son pequeñas: todas las variantes producen valores de HR@10 alrededor de 0.01, NDCG@10 entre 0.0042 y 0.0050, y MRR entre 0.0086 y 0.0100.

Además, no se observa una tendencia clara de mejora al incrementar el retorno. De hecho, los percentiles altos ($p_{75}$ y $p_{90}$) no superan de manera consistente a valores más bajos, y el retorno máximo produce un rendimiento ligeramente inferior. Esto sugiere que el `target_return` no actúa como un mecanismo de control efectivo: el modelo produce recomendaciones similares independientemente del nivel de retorno especificado.

En conjunto, estos resultados indican que el *Decision Transformer* no está utilizando de forma significativa la señal de retorno para ajustar su comportamiento, lo cual coincide con el mal comportamiento de la validación en el entrenamiento.

## 5 Conclusión

En este trabajo se implementó un sistema de recomendación basado en *Decision Transformers* sobre un entorno *offline* de calificaciones de películas, formulando el problema como un MDP y adaptando la arquitectura para incorporar información de estado, acción, return-to-go, tiempo y grupo de usuario. Además, se comparó el código de referencia provisto con una versión modificada que introduce cambios estructurales en los embeddings, la organización de la secuencia y la cabeza de predicción.

Los resultados muestran que la arquitectura propuesta logra entrenar de manera efectiva, aunque esto no se refleja en la validación durante el entrenamiento. A primera vista, este comportamiento sugiere cierto grado de sobreajuste: el modelo disminuye su pérdida de entrenamiento pero no logra mejorar la pérdida de validación, posiblemente memorizando patrones específicos de las trayectorias. En contraste, la implementación de referencia apenas reduce su pérdida y no mejora sus métricas, incluso bajo diferentes conjuntos de hiperparámetros, estrategias de *warm-up* y normalización del RTG, lo que indica limitaciones intrínsecas de su arquitectura.

A pesar del sobreajuste observado, las métricas de evaluación muestran que la versión modificada supera tanto al modelo de referencia como a un baseline de popularidad. Las mejoras absolutas son pequeñas pero sistemáticas, indicando que las modificaciones introducidas (embeddings separados, estructura intercalada y simplificación de la cabeza de predicción) aumentan la capacidad del modelo para capturar dinámicas usuario–ítem que los métodos más simples no logran representar. Una posible explicación de

la discrepancia entre la validación y el test es que la validación utiliza una función de pérdida estricta basada en cross-entropy por *timestep*, mientras que el test emplea métricas de ranking (HR@K, NDCG y MRR), más alineadas con la tarea real. Así, el modelo puede parecer sobreajustado en términos de pérdida, pero aun así mejorar en las métricas finales relevantes.

El análisis del condicionamiento por retorno muestra que variar el `target_return` entre percentiles bajos, medianos, altos y extremos no produce cambios sustanciales en la calidad de las recomendaciones. Esto sugiere que el modelo no está explotando plenamente la señal del RTG, aunque los valores más altos tienden a rendir ligeramente mejor, indicando al menos cierta sensibilidad al retorno deseado.

En general, el rendimiento absoluto de todas las variantes del modelo es bajo, lo cual es esperable dada la dificultad del problema y el escenario de *cold-start*. Además, las restricciones computacionales limitaron la exploración de arquitecturas más profundas y búsquedas más amplias de hiperparámetros. Aun así, los resultados muestran que la versión propuesta ofrece una mejora leve pero consistente sobre los métodos más simples, lo que sugiere que las ideas de diseño introducidas pueden servir como base para desarrollos futuros.

## Referencias

[1] L. Chen et al., *Decision transformer: Reinforcement learning via sequence modeling*, 2021. arXiv: 2106.01345 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2106.01345.

[2] D. C. Rajapakse and D. Leith, *Rlt4rec: Reinforcement learning transformer for user cold start and item recommendation*, 2024. arXiv: 2412.07403 [cs.IR]. [Online]. Available: https://arxiv.org/abs/2412.07403.

[3] S. Perón Santana and F. Ávila, *Decision-Transformer-Recomendaciones*, version main, Dec. 2025. [Online]. Available: https://github.com/sofips/Decision-Transformer-Recomendaciones.