

## ✓ Máster en Inteligencia Artificial Aplicada

### Unidad: Systems Recommendations - Caso Práctico 3

#### Sistema de Recomendación de Películas

Comparacion de los Framework: 'Surprise' 'Light-FM' y 'TensorFlow Recommenders'

Nombre: Patricio Galván

Fecha: 12 de MARZO 2025

### Caso Práctico Unidad 3: Sistemas de Recomendación con MovieLens 100k

#### Introducción

Los sistemas de recomendación son fundamentales en plataformas digitales para personalizar la experiencia del usuario, sugiriendo ítems relevantes como películas, productos o contenido. Este trabajo implementa y compara tres enfoques de recomendación utilizando el dataset MovieLens 100k: factorización matricial con Surprise (SVD), un modelo híbrido con LightFM (WARP) y aprendizaje profundo con TensorFlow Recommenders (TFRS). El objetivo es evaluar su rendimiento en términos de precisión (Precision@10), capacidad predictiva (AUC) y eficiencia (tiempo de entrenamiento), además de analizar su comportamiento durante el entrenamiento y la calidad de sus recomendaciones.

#### Contexto

El dataset MovieLens 100k contiene 100,000 calificaciones de 943 usuarios sobre 1,682 películas, junto con metadatos como géneros. Es un estándar en la evaluación de sistemas de recomendación por su tamaño manejable y riqueza de información. Cada modelo aborda el problema desde una perspectiva distinta:

- **Surprise (SVD):** Factorización matricial clásica, eficiente para datos explícitos.
- **LightFM (WARP):** Modelo híbrido que combina interacciones y características de ítems (géneros), optimizado para datos implícitos.

- **TFRS:** Enfoque basado en embeddings y aprendizaje profundo, diseñado para escalabilidad y personalización avanzada.

El trabajo se desarrolló en Google Colab con TensorFlow 2.15.0, TFRS 0.7.3, Surprise y LightFM, usando una CPU estándar.

## Resumen de Secciones

1. **Preparación del Entorno:** Instalación de dependencias y configuración del entorno en Colab.
2. **Preparación de Datos:** Carga y preprocesamiento de MovieLens 100k, división en entrenamiento (80%) y prueba (20%), y formato para cada framework.
3. **Implementación de Modelos:**
  - **Surprise (SVD):** Entrenado con 100 factores y 30 épocas.
  - **LightFM (WARP):** Entrenado con 32 componentes, 30 épocas y géneros como características.
  - **TFRS:** Modelo de retrieval con embeddings de 64 dimensiones, 30 épocas (mejorado tras iteraciones iniciales).
4. **Comparación de Resultados:** Tabla y gráficos con métricas y curvas de entrenamiento.
5. **Ejemplos Concretos:** Recomendaciones y predicciones para un usuario específico (usuario 196).

## Resultados

Los resultados finales tras entrenar y evaluar los modelos durante 30 épocas son:

Modelo	Precision@10	AUC	Tiempo de Entrenamiento (s)
Surprise (SVD)	0.5719	0.7714	2.66
LightFM (WARP)	0.3474	0.9355	21.79
TFRS	0.3652	0.7487	358.34

## Análisis de Métricas y Gráficos

### Análisis de Gráficos

- **Precision@10 por Modelo:** El gráfico de barras muestra que Surprise lidera con un valor de 0.5719, seguido por TFRS (0.3652) y LightFM (0.3474). Esto evidencia una clara superioridad de Surprise en la relevancia de sus recomendaciones.
- **AUC por Modelo:** LightFM destaca con un AUC de 0.9355, seguido por Surprise (0.7714) y TFRS (0.7487). El gráfico de barras refleja esta tendencia, con LightFM en la cima, indicando su capacidad para rankear ítems.
- **Tiempo de Entrenamiento:** TFRS presenta el mayor tiempo (358.34s), seguido por LightFM (21.79s) y Surprise (2.66s). El gráfico de barras resalta la diferencia significativa de TFRS, lo que subraya su costo computacional.

- **Curvas de Entrenamiento:** La gráfica de pérdida (RMSE para Surprise, pérdida personalizada para LightFM y TFRS) por época muestra lo siguiente:
  - **Surprise:** Converge rápidamente, alcanzando un RMSE estable cerca de 0.87 tras 30 épocas, lo que indica un ajuste eficiente.
  - **LightFM:** Se estabiliza después de unas 10-15 épocas, con una convergencia más lenta pero efectiva, reflejando el impacto de WARP en optimizar el ranking.
  - **TFRS:** Muestra una convergencia más lenta, con una pérdida final más alta que los otros modelos, lo que sugiere menor eficiencia y necesidad de optimización adicional.

## Análisis Detallado de Métricas

- **Surprise (SVD):**
  - **Precision@10 (0.5719):** Este valor indica que aproximadamente el 57% de las 10 películas recomendadas para un usuario son relevantes (rating  $\geq 4.0$ ), posicionándolo como el mejor en precisión. Esto se alinea con su enfoque en optimizar ratings explícitos mediante factorización matricial.
  - **AUC (0.7714):** Representa una capacidad moderada para distinguir entre ítems relevantes e irrelevantes, inferior a LightFM pero suficiente para un modelo basado en datos explícitos.
  - **Tiempo de Entrenamiento (2.66s):** Es el más rápido, reflejando su eficiencia computacional, ideal para entornos con recursos limitados.
  - **Curva de Entrenamiento:** La caída pronunciada en RMSE sugiere una convergencia rápida, alcanzando un valor estable, lo que indica un buen ajuste con pocos recursos.
- **LightFM (WARP):**
  - **Precision@10 (0.3474):** Aunque menor que Surprise, este valor muestra que cerca del 35% de las recomendaciones son relevantes. Su enfoque en datos implícitos y géneros explica una precisión más baja en un contexto de ratings explícitos.
  - **AUC (0.9355):** El valor más alto entre los modelos, indicando una excelente capacidad para rankear ítems correctamente, especialmente en escenarios implícitos donde los géneros juegan un papel clave.
  - **Tiempo de Entrenamiento (21.79s):** Moderado, significativamente menor que TFRS, lo que lo hace viable para sistemas con metadatos sin exigir demasiados recursos.
  - **Curva de Entrenamiento:** La pérdida se estabiliza después de unas 10-15 épocas, con una convergencia más lenta que Surprise pero efectiva, reflejando el impacto de WARP en optimizar el ranking.
- **TFRS:**
  - **Precision@10 (0.3652):** Una mejora notable respecto a su versión inicial (0.0155), alcanzando un 36.5% de relevancia. Esto sugiere que las 30 épocas y los ajustes en

embeddings (64 dimensiones) mejoraron su capacidad predictiva, aunque sigue por debajo de Surprise.

- **AUC (0.7487):** El valor más bajo, indicando una menor habilidad para distinguir ítems relevantes, posiblemente por la falta de características adicionales como géneros en el modelo básico de retrieval.
- **Tiempo de Entrenamiento (358.34s):** El más alto, casi 17 veces mayor que LightFM y 135 veces mayor que Surprise, lo que lo hace poco práctico en un entorno de CPU sin optimización adicional (e.g., GPU).
- **Curva de Entrenamiento:** La pérdida disminuye lentamente, con un valor final más alto que los otros modelos, lo que indica una convergencia menos eficiente y sugiere que el modelo podría beneficiarse de más ajuste o datos.

## Comparación y Tendencias

- **Precision@10:** Surprise lidera claramente (0.5719), seguido por TFRS (0.3652) y LightFM (0.3474). Esto refleja que Surprise es más efectivo para recomendar ítems relevantes basándose en ratings explícitos, mientras que LightFM y TFRS, orientados a ranking implícito, tienen un desempeño inferior en este contexto.
- **AUC:** LightFM domina (0.9355), destacando su capacidad para rankear ítems, seguido por Surprise (0.7714) y TFRS (0.7487). Esto sugiere que LightFM es superior en distinguir preferencias implícitas, mientras que TFRS necesita mejoras.
- **Tiempo de Entrenamiento:** Surprise (2.66s) es el más eficiente, seguido por LightFM (21.79s), mientras que TFRS (358.34s) es desproporcionadamente lento, limitando su practicidad sin hardware especializado.
- **Curvas de Entrenamiento:** Surprise converge rápido, LightFM muestra un equilibrio, y TFRS tiene una convergencia más lenta y menos óptima, lo que respalda la necesidad de optimización adicional.

## Análisis de Recomendaciones para el Usuario 196

Se analizaron las recomendaciones y las predicciones de películas vistas para el usuario 196, destacando diferencias en el enfoque de cada modelo:

- **Surprise (SVD):**
  - **Películas Vistas:** Predice ratings altos y consistentes con las valoraciones reales del usuario (e.g., "Secrets & Lies" 5.0 -> 4.38, "English Patient" 5.0 -> 4.35), mostrando una reconstrucción precisa de preferencias explícitas.
  - **Recomendaciones:** Sugiere películas populares y bien valoradas como "Raiders of the Lost Ark" (4.75) y "Braveheart" (4.67), alineándose con su alta Precision@10 (0.5719). Esto refleja su capacidad para identificar ítems relevantes basados en patrones globales de ratings.
  - **Fuerza:** Su bajo tiempo de entrenamiento (2.66s) y alta precisión lo hacen ideal para sistemas rápidos y efectivos con datos explícitos.

- **LightFM (WARP):**

- **Películas Vistas:** Los scores no reflejan bien los ratings reales (e.g., "Men in Black" 2.0 -> 0.0798, "English Patient" 5.0 -> -0.1735), ya que LightFM está diseñado para ranking implícito, no para predecir ratings absolutos.
- **Recomendaciones:** Propone películas de acción y suspenso como "Scream" (1.3105) y "Volcano" (1.2535), influenciadas por géneros y datos implícitos, pero con menor Precision@10 (0.3474). Su alto AUC (0.9355) indica una excelente capacidad para distinguir ítems relevantes de no relevantes en un contexto implícito.
- **Fuerza:** Es adecuado para sistemas con metadatos (géneros) y datos implícitos, con un tiempo razonable (21.79s).

- **TFRS:**

- **Películas Vistas:** Mejora respecto a versiones iniciales, rankeando "Secrets & Lies" (5.0 -> 1.4179) como la mejor, pero incluye ítems de menor rating en el top-5 (e.g., "Mighty Aphrodite" 2.0 -> 1.1776), mostrando una alineación parcial con las preferencias.
- **Recomendaciones:** Sugiere películas dramáticas/románticas como "Antonia's Line" (1.3161) y "Sense and Sensibility" (1.2720), con una Precision@10 (0.3652) superior a la inicial (0.0155), pero aún por debajo de Surprise. Su AUC (0.7487) es el más bajo, indicando menor capacidad discriminativa.
- **Limitación:** El tiempo de entrenamiento (358.34s) es excesivo para su desempeño, sugiriendo que el modelo básico de retrieval requiere más optimización (e.g., géneros, tarea de ranking).

## Conclusión General

- **Surprise (SVD)** se destaca como el mejor modelo para este caso, con la mayor Precision@10 (0.5719), un AUC sólido (0.7714) y el menor tiempo de entrenamiento (2.66s). Sus recomendaciones son intuitivas y reflejan bien las preferencias explícitas del usuario, siendo ideal para sistemas simples y eficientes con MovieLens 100k.
- **LightFM (WARP)** sobresale en AUC (0.9355), mostrando su fortaleza en datos implícitos y metadatos, aunque su Precision@10 (0.3474) y las recomendaciones menos populares sugieren que requiere más ajuste para capturar gustos específicos.
- **TFRS**, con 30 épocas, mejora significativamente su Precision@10 (0.3652) respecto a la versión inicial (0.0155), pero su AUC (0.7487) y alto costo computacional (358.34s) lo hacen menos competitivo. Sus recomendaciones, aunque diversas, no se alinean tan bien con las preferencias del usuario, indicando que necesita características adicionales (e.g., géneros) o un enfoque combinado (retrieval + ranking).

## Recomendaciones Finales

- Para un sistema de recomendación inmediato y eficiente, **Surprise (SVD)** es la mejor opción, ofreciendo el mejor balance entre precisión y velocidad.
- Si se priorizan datos implícitos y metadatos, **LightFM** ofrece un balance adecuado, ajustable con más épocas o hiperparámetros para mejorar su precisión.
- **TFRS** tiene potencial para datasets más grandes o con GPU, pero requiere optimización adicional (e.g., integración de géneros, ajuste de hiperparámetros, o hardware especializado) para justificar su costo computacional en este contexto.

=====CODIGO=====

## ✓ 1.0 Configuración del Entorno

```
# =====
# 1. Configuración del Entorno
# =====

# Desinstalar versiones conflictivas (solo si es necesario, comentado por ahora)
# !pip uninstall -y tensorflow tensorflow-recommenders tf-keras tensorflow-text tensorsto

# Instalar versiones específicas y compatibles
!pip install tensorflow==2.15.0 tensorflow-recommenders==0.7.3 scikit-surprise lightfm -q
!pip install seaborn -q
```



```
154.4/154.4 kB 3.0 MB/s eta 0:00:00
Installing build dependencies ... done
Getting requirements to build wheel ... done
Preparing metadata (pyproject.toml) ... done
316.4/316.4 kB 10.9 MB/s eta 0:00:00
Preparing metadata (setup.py) ... done
475.3/475.3 MB 2.6 MB/s eta 0:00:00
96.2/96.2 kB 6.4 MB/s eta 0:00:00
1.7/1.7 MB 62.1 MB/s eta 0:00:00
1.0/1.0 MB 46.9 MB/s eta 0:00:00
5.5/5.5 MB 92.9 MB/s eta 0:00:00
442.0/442.0 kB 23.5 MB/s eta 0:00:00
78.4/78.4 kB 5.4 MB/s eta 0:00:00
Building wheel for scikit-surprise (pyproject.toml) ... done
Building wheel for lightfm (setup.py) ... done
```

```
ERROR: pip's dependency resolver does not currently take into account all the package
jax 0.5.2 requires ml_dtypes>=0.4.0, but you have ml-dtypes 0.2.0 which is incompatib
tf-keras 2.18.0 requires tensorflow<2.19,>=2.18, but you have tensorflow 2.15.0 which
tensorflow-text 2.18.1 requires tensorflow<2.19,>=2.18.0, but you have tensorflow 2.1
tensorstore 0.1.72 requires ml_dtypes>=0.3.1, but you have ml-dtypes 0.2.0 which is i
```

```
# Importar librerías necesarias
import os
import time
import pandas as pd
```

```

import numpy as np
import matplotlib.pyplot as plt

import tensorflow as tf
import tensorflow_recommenders as tfrs

from sklearn.metrics import roc_auc_score
from sklearn.model_selection import train_test_split
from scipy.sparse import coo_matrix
from surprise import Dataset, Reader, SVD
from surprise.model_selection import cross_validate
from surprise import accuracy

from lightfm import LightFM
from lightfm.data import Dataset as LFM_Dataset
from lightfm.evaluation import precision_at_k as lfm_precision_at_k, auc_score as lfm_auc

from google.colab import drive

# Montar Google Drive
drive.mount('/content/drive')

# Verificar versiones
print( "="*27)
print("Verificacion de Versiones")
print( "="*27)
print("\nNumPy version:", np.__version__)
print("TensorFlow version:", tf.__version__)
print("TFRS version:", tfrs.__version__)

# Verificar disponibilidad de GPU (aunque usemos CPU por ahora)
print("Num GPUs Available:", len(tf.config.experimental.list_physical_devices('GPU')))
```

 Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.r

```

=====
Verificacion de Versiones
=====
```

```

NumPy version: 1.26.4
TensorFlow version: 2.15.0
TFRS version: v0.7.3
Num GPUs Available: 0
```

## ✓ 2.0 Preparación de Datos

```

# =====
# 2. Preparación de Datos
# =====

print("\n" + "="*50)
print("Preparación de Datos para Sistemas de Recomendación")
print("="*50)
```

```

# Definición de Paths del Dataset
print("\nDefinición de rutas del dataset:")
DATA_PATH = "/content/drive/MyDrive/Colab Notebooks/Recomendation Systems/Caso 3/ml-100k"
FILE_RATINGS = os.path.join(DATA_PATH, "u.data")
FILE_MOVIES = os.path.join(DATA_PATH, "u.item")
FILE_USERS = os.path.join(DATA_PATH, "u.user")
print(f" - Ratings: {FILE_RATINGS}")
print(f" - Movies: {FILE_MOVIES}")
print(f" - Users: {FILE_USERS}")

# Carga de Datos en DataFrames
print("\nCargando datos en DataFrames...")

## 2.1 Cargar Ratings
ratings_columns = ["user_id", "movie_id", "rating", "timestamp"]
ratings_df = pd.read_csv(FILE_RATINGS, sep='\t', names=ratings_columns, encoding="ISO-885
ratings_df["user_id"] = ratings_df["user_id"].astype(int)
ratings_df["movie_id"] = ratings_df["movie_id"].astype(int)

## 2.2 Cargar Información de Películas
movies_columns = [
    "movie_id", "title", "release_date", "video_release_date", "IMDb_URL",
    "unknown", "Action", "Adventure", "Animation", "Children", "Comedy", "Crime",
    "Documentary", "Drama", "Fantasy", "Film-Noir", "Horror", "Musical",
    "Mystery", "Romance", "Sci-Fi", "Thriller", "War", "Western"
]
movies_df = pd.read_csv(FILE_MOVIES, sep='|', names=movies_columns, encoding="ISO-8859-1"
                        usecols=[0, 1] + list(range(5, 24)))

## 2.3 Cargar Información de Usuarios
users_columns = ["user_id", "age", "gender", "occupation", "zip_code"]
users_df = pd.read_csv(FILE_USERS, sep='|', names=users_columns, encoding="ISO-8859-1")

# Validación Básica de los Datos Cargados
print("\nValidación inicial de los datos:")
print(" - Ratings Dataset (primeras 3 filas):")
print(ratings_df.head(3).to_string(index=False))
print(" - Movies Dataset (primeras 3 filas):")
print(movies_df.head(3).to_string(index=False))
print(" - Users Dataset (primeras 3 filas):")
print(users_df.head(3).to_string(index=False))

# Estadísticas Rápidas
print("\nEstadísticas de los DataFrames:")
print(f" - Ratings: {ratings_df.shape[0]} filas, {ratings_df.shape[1]} columnas")
print(f" - Movies: {movies_df.shape[0]} filas, {movies_df.shape[1]} columnas")
print(f" - Users: {users_df.shape[0]} filas, {users_df.shape[1]} columnas")
print(f" - Usuarios únicos: {ratings_df['user_id'].nunique()}")
print(f" - Películas únicas: {ratings_df['movie_id'].nunique()}")

# Tratamiento de Datos
print("\nTratamiento de datos (eliminando nulos y verificando consistencia)...")
ratings_df = ratings_df.dropna()
movies_df = movies_df.dropna()

```



```

users_df = users_df.dropna()
ratings_df = ratings_df[ratings_df["movie_id"].isin(movies_df["movie_id"])]
ratings_df = ratings_df[ratings_df["user_id"].isin(users_df["user_id"])]

# División en Conjuntos de Entrenamiento y Prueba
print("\nDividiendo datos en entrenamiento (80%) y prueba (20%)...")
train_ratings, test_ratings = train_test_split(ratings_df, test_size=0.2, random_state=42)
print(f" - Entrenamiento: {train_ratings.shape[0]} interacciones")
print(f" - Prueba: {test_ratings.shape[0]} interacciones")

# Preparación de Datos para los 3 Frameworks
print("\nPreparando datos para los frameworks...")

## 2.1 Formato para Surprise
print(" - Formato para Surprise:")
reader = Reader(rating_scale=(1, 5))
train_surprise = Dataset.load_from_df(train_ratings[['user_id', 'movie_id', 'rating']], r
test_surprise = list(test_ratings[['user_id', 'movie_id', 'rating']].itertuples(index=False,

## 2.2 Formato para LightFM
print(" - Formato para LightFM:")
lfm_dataset = LFM_Dataset()
lfm_dataset.fit(users=ratings_df["user_id"].unique(), items=ratings_df["movie_id"].unique
train_interactions, _ = lfm_dataset.build_interactions(train_ratings[['user_id', 'movie_id', 'rating']],
test_interactions, _ = lfm_dataset.build_interactions(test_ratings[['user_id', 'movie_id', 'rating']],

## 2.3 Formato para TensorFlow Recommenders (TFRS)
print(" - Formato para TensorFlow Recommenders (TFRS):")
train_tf = tf.data.Dataset.from_tensor_slices({
    "user_id": train_ratings["user_id"].astype(str).values,
    "movie_id": train_ratings["movie_id"].astype(str).values,
    "rating": train_ratings["rating"].values
})
test_tf = tf.data.Dataset.from_tensor_slices({
    "user_id": test_ratings["user_id"].astype(str).values,
    "movie_id": test_ratings["movie_id"].astype(str).values,
    "rating": test_ratings["rating"].values
})
train_tf = train_tf.map(lambda x: {"user_id": x["user_id"], "movie_id": x["movie_id"]})
test_tf = test_tf.map(lambda x: {"user_id": x["user_id"], "movie_id": x["movie_id"]})

# Validación de Formatos
print("\nValidación final de los formatos preparados:")
print(" - Datos para Surprise:")
print(f"    Tipo: {type(train_surprise)}")
print(f"    Ejemplo de test (primeras 3 tuplas): {test_surprise[:3]}")
print(" - Datos para LightFM:")
print(f"    Shape Train Interactions: {train_interactions.shape}")
print(f"    Shape Test Interactions: {test_interactions.shape}")
print(" - Datos para TFRS:")
print(f"    Ejemplo de datos en TF (primer elemento): {next(iter(train_tf))}")

print("\n")

```



```
=====
Preparación de Datos para Sistemas de Recomendación
=====
```

Definición de rutas del dataset:

- Ratings: /content/drive/MyDrive/Colab Notebooks/Recommendation Systems/Caso 3/ml-
- Movies: /content/drive/MyDrive/Colab Notebooks/Recommendation Systems/Caso 3/ml-1
- Users: /content/drive/MyDrive/Colab Notebooks/Recommendation Systems/Caso 3/ml-10

Cargando datos en DataFrames...

Validación inicial de los datos:

- Ratings Dataset (primeras 3 filas):

user_id	movie_id	rating	timestamp
196	242	3	881250949
186	302	3	891717742
22	377	1	878887116

- Movies Dataset (primeras 3 filas):

movie_id	title	unknown	Action	Adventure	Animation	Children	Comedy
1	Toy Story (1995)	0	0	0	1	1	
2	GoldenEye (1995)	0	1	1	0	0	
3	Four Rooms (1995)	0	0	0	0	0	

- Users Dataset (primeras 3 filas):

user_id	age	gender	occupation	zip_code
1	24	M	technician	85711
2	53	F	other	94043
3	23	M	writer	32067

Estadísticas de los DataFrames:

- Ratings: 100000 filas, 4 columnas
- Movies: 1682 filas, 21 columnas
- Users: 943 filas, 5 columnas
- Usuarios únicos: 943
- Películas únicas: 1682

Tratamiento de datos (eliminando nulos y verificando consistencia)...

Dividiendo datos en entrenamiento (80%) y prueba (20%)...

- Entrenamiento: 80000 interacciones
- Prueba: 20000 interacciones

Preparando datos para los frameworks...

- Formato para Surprise:
- Formato para LightFM:
- Formato para TensorFlow Recommenders (TFRS):

Validación final de los formatos preparados:

- Datos para Surprise:

Tipo: <class 'surprise.dataset.DatasetAutoFolds'>

Ejemplo de test (primeras 3 tuplas): [(877, 381, 4), (815, 602, 3), (94, 431, 4)]

- Datos para LightFM:

Shape Train Interactions: (943, 1682)

Shape Test Interactions: (943, 1682)

- Datos para TFRS:

Ejemplo de datos en TF (primer elemento): {'user\_id': <tf.Tensor: shape=(), dtype=

```
=====
```

## ✓ 3. Implementación de Modelos


```
# =====
# 3. Implementación de Modelos
# =====
print( "="*35)
print("Configuración de Hiperparámetros")
print( "="*35)
# Hiperparámetros para Surprise (SVD)
SURPRISE_N_FACTORS = 100      # Número de factores latentes para la factorización matricial
SURPRISE_N_EPOCHS = 30       # Número de épocas de entrenamiento

# Hiperparámetros para LightFM (WARP)
LIGHTFM_NO_COMPONENTS = 32    # Dimensión de los embeddings para usuarios e ítems. 32 es ef
LIGHTFM_EPOCHS = 30           # Número de épocas de entrenamiento. 30 es un buen balance par
LIGHTFM_MAX_SAMPLED = 10      # Máximo de muestras negativas por actualización en WARP. 10 a
LIGHTFM_NUM_THREADS = 2       # Hilos para paralelizar en CPU. 2 es conservador para Colab g
LIGHTFM_LEARNING_RATE = 0.05  # Tasa de aprendizaje para el optimizador. 0.05 es estándar
LIGHTFM_LOSS = 'warp'         # Función de pérdida. WARP es ideal para datos implícitos como
# Explicación: 32 componentes es eficiente en CPU y suficiente para 100k; 30 épocas optim
# que es adecuado para rankings implícitos; max_sampled=10 acelera el entrenamiento.

# Hiperparámetros para TensorFlow Recommenders (TFRS)
TFRS_EMBEDDING_DIM = 64       # Aumentado para mayor capacidad.
TFRS_EPOCHS = 30              # Más épocas para mejor convergencia.
TFRS_BATCH_SIZE = 512         # Reducido para menor carga en CPU.
TFRS_LEARNING_RATE = 0.01     # Menor para estabilidad.

print(f"\nSurprise: n_factors={SURPRISE_N_FACTORS}, n_epochs={SURPRISE_N_EPOCHS}\n")
print(f"LightFM: no_components={LIGHTFM_NO_COMPONENTS}, epochs={LIGHTFM_EPOCHS}, max_samp
print(f"TFRS: embedding_dim={TFRS_EMBEDDING_DIM}, epochs={TFRS_EPOCHS}, batch_size={TFRS_

# Diccionarios para almacenar métricas y tiempos
results = {
    "Surprise": {"Precision@10": 0, "AUC": 0, "Train Time": 0},
    "LightFM": {"Precision@10": 0, "AUC": 0, "Train Time": 0},
    "TFRS": {"Precision@10": 0, "AUC": 0, "Train Time": 0}
}
loss_history = {"Surprise": [], "LightFM": [], "TFRS": []}
```

 =====  
Configuración de Hiperparámetros  
=====

```
Surprise: n_factors=100, n_epochs=30

LightFM: no_components=32, epochs=30, max_sampled=10, num_threads=2

TFRS: embedding_dim=64, epochs=30, batch_size=512, learning_rate=0.01
```

## ✓ 3.1 Surprise (SVD)

```
# -----
# 3.1 Surprise (SVD)
# -----

# Encabezado de sección
print("\n" + "="*50)
print("Entrenamiento y Evaluación de Surprise (SVD)")
print("="*50)

# Inicio del entrenamiento
print("Iniciando entrenamiento...")
start_time = time.time()

# Configurar y entrenar el modelo
print("Configurando modelo SVD...")
svd_model = SVD(n_factors=SURPRISE_N_FACTORS, n_epochs=SURPRISE_N_EPOCHS, verbose=False)
trainset = train_surprise.build_full_trainset()
print(f"Entrenando SVD con {SURPRISE_N_FACTORS} factores y {SURPRISE_N_EPOCHS} épocas...")
svd_model.fit(trainset)

# Registrar tiempo de entrenamiento
results["Surprise"]["Train Time"] = time.time() - start_time

# Generar predicciones y calcular métricas
print("\nGenerando predicciones para el conjunto de prueba...")
test_predictions = [svd_model.predict(uid, iid, r_ui) for (uid, iid, r_ui) in test_surpri

def precision_at_k_surprise(predictions, k=10, threshold=4.0):
    user_pred = {}
    for pred in predictions:
        user_pred.setdefault(pred.uid, []).append((pred.est, pred.iid, pred.r_ui))
    precision = 0
    for uid, preds in user_pred.items():
        preds.sort(reverse=True)
        top_k = preds[:k]
        relevant = sum(1 for est, _, r_ui in top_k if r_ui >= threshold)
        precision += relevant / k
    return precision / len(user_pred)

def auc_score_manual(predictions, threshold=4.0):
    y_true = [1 if pred.r_ui >= threshold else 0 for pred in predictions]
    y_score = [pred.est for pred in predictions]
    return roc_auc_score(y_true, y_score)

# Calcular métricas
print("Calculando métricas...")
results["Surprise"]["Precision@10"] = precision_at_k_surprise(test_predictions)
results["Surprise"]["AUC"] = auc_score_manual(test_predictions)

# Imprimir resultados principales
print("="*50)
```

```

print("Resultados de Surprise (SVD):")
print(f" - Precision@10: {results['Surprise']['Precision@10']:.4f}")
print(f" - AUC: {results['Surprise']['AUC']:.4f}")
print(f" - Tiempo de entrenamiento: {results['Surprise']['Train Time']:.2f} segundos")
print("="*50)

# Calcular pérdida por época (RMSE en validación)
print("\nCalculando pérdida por época (RMSE en validación):")
from surprise.model_selection import train_test_split as surprise_split
train_temp, val_temp = surprise_split(train_surprise, test_size=0.2)
svd_temp = SVD(n_factors=SURPRISE_N_FACTORS, n_epochs=SURPRISE_N_EPOCHS, verbose=False)
loss_history["Surprise"] = []

for epoch in range(SURPRISE_N_EPOCHS):
    svd_temp.n_epochs = epoch + 1
    svd_temp.fit(train_temp)
    val_preds = svd_temp.test(val_temp)
    rmse = accuracy.rmse(val_preds, verbose=False)
    loss_history["Surprise"].append(rmse)
    print(f" - Época {epoch+1:2d}/{SURPRISE_N_EPOCHS} | RMSE: {rmse:.4f}")

```



```

=====
Entrenamiento y Evaluación de Surprise (SVD)
=====
Iniciando entrenamiento...
Configurando modelo SVD...
Entrenando SVD con 100 factores y 30 épocas...

Generando predicciones para el conjunto de prueba...
Calculando métricas...
=====
Resultados de Surprise (SVD):
- Precision@10: 0.5719
- AUC: 0.7714
- Tiempo de entrenamiento: 2.66 segundos
=====

Calculando pérdida por época (RMSE en validación):
- Época 1/30 | RMSE: 1.0191
- Época 2/30 | RMSE: 0.9903
- Época 3/30 | RMSE: 0.9763
- Época 4/30 | RMSE: 0.9699
- Época 5/30 | RMSE: 0.9636
- Época 6/30 | RMSE: 0.9604
- Época 7/30 | RMSE: 0.9579
- Época 8/30 | RMSE: 0.9548
- Época 9/30 | RMSE: 0.9536
- Época 10/30 | RMSE: 0.9515
- Época 11/30 | RMSE: 0.9499
- Época 12/30 | RMSE: 0.9514
- Época 13/30 | RMSE: 0.9494
- Época 14/30 | RMSE: 0.9464
- Época 15/30 | RMSE: 0.9461
- Época 16/30 | RMSE: 0.9471
- Época 17/30 | RMSE: 0.9445
- Época 18/30 | RMSE: 0.9439
- Época 19/30 | RMSE: 0.9444

```

- Época 20/30 | RMSE: 0.9424
- Época 21/30 | RMSE: 0.9432
- Época 22/30 | RMSE: 0.9431
- Época 23/30 | RMSE: 0.9440
- Época 24/30 | RMSE: 0.9480
- Época 25/30 | RMSE: 0.9487
- Época 26/30 | RMSE: 0.9439
- Época 27/30 | RMSE: 0.9475
- Época 28/30 | RMSE: 0.9476
- Época 29/30 | RMSE: 0.9463
- Época 30/30 | RMSE: 0.9482

## ✓ 3.2 LightFM (WARP)

```
# -----
# 3.2 LightFM (WARP)
# -----

# Encabezado de sección
print("\n" + "="*50)
print("Entrenamiento y Evaluación de LightFM (WARP)")
print("="*50)

# Inicio del entrenamiento
print("Iniciando entrenamiento...")
start_time = time.time()

# Preparar dataset con características (usuarios, ítems y géneros)
lfm_dataset = LFM_Dataset()
lfm_dataset.fit(
    users=ratings_df["user_id"].unique(),
    items=ratings_df["movie_id"].unique(),
    item_features=movies_df.columns[2:] # Géneros desde 'unknown' hasta 'Western'
)

# Construir características de ítems con géneros
print("Construyendo características de ítems con géneros...")
genre_columns = movies_df.columns[2:] # Lista de nombres de géneros
item_features_data = [
    (row["movie_id"], {genre: row[genre] for genre in genre_columns if row[genre] == 1})
    for _, row in movies_df.iterrows()
]
item_features = lfm_dataset.build_item_features(item_features_data)

# Construir interacciones completas
print("Construyendo matrices de interacciones...")
train_interactions, _ = lfm_dataset.build_interactions(train_ratings[['user_id', 'movie_id']])
test_interactions, _ = lfm_dataset.build_interactions(test_ratings[['user_id', 'movie_id']])

# Inicializar el modelo LightFM
print("Inicializando modelo LightFM...")
lfm_model = LightFM(
    no_components=LIGHTFM_NO_COMPONENTS,
```

```

    loss=LIGHTFM_LOSS,
    learning_rate=LIGHTFM_LEARNING_RATE,
    max_sampled=LIGHTFM_MAX_SAMPLED
)

# Entrenamiento con pérdida por época
print("\nEntrenamiento por épocas:")
loss_history["LightFM"] = []
for epoch in range(LIGHTFM_EPOCHS):
    lfm_model.fit_partial(
        interactions=train_interactions,
        item_features=item_features,
        epochs=1,
        num_threads=LIGHTFM_NUM_THREADS,
        verbose=False
    )
    auc_epoch = lfm_auc_score(
        lfm_model,
        train_interactions, # Usamos train como proxy de validación por simplicidad
        item_features=item_features,
        num_threads=LIGHTFM_NUM_THREADS
    ).mean()
    loss_history["LightFM"].append(auc_epoch)
    print(f" - Época {epoch+1:2d}/{LIGHTFM_EPOCHS} | AUC en entrenamiento: {auc_epoch:.4f}")

# Registrar tiempo de entrenamiento
train_time = time.time() - start_time
results["LightFM"]["Train Time"] = train_time

# Evaluación en el conjunto de prueba
print("\nEvaluación en conjunto de prueba:")
precision_at_10 = lfm_precision_at_k(
    lfm_model,
    test_interactions,
    train_interactions=train_interactions,
    item_features=item_features,
    k=10,
    num_threads=LIGHTFM_NUM_THREADS
).mean()
results["LightFM"]["Precision@10"] = precision_at_10

auc_test = lfm_auc_score(
    lfm_model,
    test_interactions,
    train_interactions=train_interactions,
    item_features=item_features,
    num_threads=LIGHTFM_NUM_THREADS
).mean()
results["LightFM"]["AUC"] = auc_test

# Imprimir resultados finales
print("\n"*50)
print("Resultados de LightFM:")
print(f" - Precision@10: {results['LightFM']['Precision@10']:.4f}")
print(f" - AUC: {results['LightFM']['AUC']:.4f}")

```

```
print(f" - Tiempo de entrenamiento: {results['LightFM']['Train Time']:.2f} segundos")
print("="*50)
```



```
=====
Entrenamiento y Evaluación de LightFM (WARP)
=====
Iniciando entrenamiento...
Construyendo características de ítems con géneros...
Construyendo matrices de interacciones...
Inicializando modelo LightFM...
```

Entrenamiento por épocas:

```
- Época 1/30 | AUC en entrenamiento: 0.8595
- Época 2/30 | AUC en entrenamiento: 0.8850
- Época 3/30 | AUC en entrenamiento: 0.8981
- Época 4/30 | AUC en entrenamiento: 0.9076
- Época 5/30 | AUC en entrenamiento: 0.9150
- Época 6/30 | AUC en entrenamiento: 0.9207
- Época 7/30 | AUC en entrenamiento: 0.9252
- Época 8/30 | AUC en entrenamiento: 0.9287
- Época 9/30 | AUC en entrenamiento: 0.9313
- Época 10/30 | AUC en entrenamiento: 0.9335
- Época 11/30 | AUC en entrenamiento: 0.9353
- Época 12/30 | AUC en entrenamiento: 0.9369
- Época 13/30 | AUC en entrenamiento: 0.9382
- Época 14/30 | AUC en entrenamiento: 0.9393
- Época 15/30 | AUC en entrenamiento: 0.9403
- Época 16/30 | AUC en entrenamiento: 0.9412
- Época 17/30 | AUC en entrenamiento: 0.9420
- Época 18/30 | AUC en entrenamiento: 0.9427
- Época 19/30 | AUC en entrenamiento: 0.9433
- Época 20/30 | AUC en entrenamiento: 0.9440
- Época 21/30 | AUC en entrenamiento: 0.9446
- Época 22/30 | AUC en entrenamiento: 0.9452
- Época 23/30 | AUC en entrenamiento: 0.9457
- Época 24/30 | AUC en entrenamiento: 0.9462
- Época 25/30 | AUC en entrenamiento: 0.9467
- Época 26/30 | AUC en entrenamiento: 0.9472
- Época 27/30 | AUC en entrenamiento: 0.9476
- Época 28/30 | AUC en entrenamiento: 0.9480
- Época 29/30 | AUC en entrenamiento: 0.9484
- Época 30/30 | AUC en entrenamiento: 0.9487
```

Evaluación en conjunto de prueba:

```
=====
Resultados de LightFM:
- Precision@10: 0.3474
- AUC: 0.9355
- Tiempo de entrenamiento: 21.79 segundos
=====
```

## ✓ 3.3 TensorFlow Recommenders (TFRS)

```
# -----
# 3.3 TensorFlow Recommenders (TFRS)
```



```

# -----

# Encabezado de sección
print("\n" + "="*50)
print("Entrenamiento y Evaluación de TensorFlow Recommenders (TFRS) - Mejorado")
print("="*50)

# Inicio del entrenamiento
print("Iniciando entrenamiento...")
start_time = time.time()

# Convertir IDs a strings (ya hecho previamente)
ratings_df["user_id_str"] = ratings_df["user_id"].astype(str)
ratings_df["movie_id_str"] = ratings_df["movie_id"].astype(str)
train_ratings["user_id_str"] = train_ratings["user_id"].astype(str)
train_ratings["movie_id_str"] = train_ratings["movie_id"].astype(str)
test_ratings["user_id_str"] = test_ratings["user_id"].astype(str)
test_ratings["movie_id_str"] = test_ratings["movie_id"].astype(str)

# Definir modelo TFRS mejorado
class MovieLensModel(tfrs.Model):
    def __init__(self):
        super().__init__()
        # Embeddings para usuarios
        self.user_embedding = tf.keras.Sequential([
            tf.keras.layers.StringLookup(vocabulary=ratings_df["user_id_str"].unique(), m
            tf.keras.layers.Embedding(len(ratings_df["user_id_str"].unique()) + 1, TFRS_E
        ])
        # Embeddings para ítems
        self.item_embedding = tf.keras.Sequential([
            tf.keras.layers.StringLookup(vocabulary=ratings_df["movie_id_str"].unique(),
            tf.keras.layers.Embedding(len(ratings_df["movie_id_str"].unique()) + 1, TFRS_
        ])
        # Tarea de retrieval con candidatos optimizados (top 100 ítems más populares)
        popular_items = ratings_df["movie_id_str"].value_counts().index[:100].tolist()
        self.task = tfrs.tasks.Retrieval(
            metrics=tfrs.metrics.FactorizedTopK(
                candidates=tf.data.Dataset.from_tensor_slices(popular_items).batch(128).re
            )
        )

    def compute_loss(self, features, training=False):
        user_embeddings = self.user_embedding(features["user_id_str"])
        item_embeddings = self.item_embedding(features["movie_id_str"])
        return self.task(user_embeddings, item_embeddings)

# Preparar datos
print("Preparando datos para TFRS...")
train_tf = tf.data.Dataset.from_tensor_slices(
    {"user_id_str": train_ratings["user_id_str"].values, "movie_id_str": train_ratings["r
").batch(TFRS_BATCH_SIZE)
test_tf = tf.data.Dataset.from_tensor_slices(
    {"user_id_str": test_ratings["user_id_str"].values, "movie_id_str": test_ratings["mov
).batch(TFRS_BATCH_SIZE)

```

```

# Inicializar y compilar modelo
print("Iniciando modelo TFRS...")
tfrs_model = MovieLensModel()
tfrs_model.compile(optimizer=tf.keras.optimizers.Adagrad(learning_rate=TFRS_LEARNING_RATE)

# Entrenamiento
print("\nEntrenamiento por épocas:")
history = tfrs_model.fit(
    train_tf,
    epochs=TFRS_EPOCHS,
    verbose=0
)
loss_history["TFRS"] = history.history["total_loss"]
for epoch, loss in enumerate(loss_history["TFRS"], 1):
    print(f" - Época {epoch:2d}/{TFRS_EPOCHS} | Pérdida: {loss:.4f}")

# Registrar tiempo
results["TFRS"]["Train Time"] = time.time() - start_time

# Evaluación
print("\nEvaluación en conjunto de prueba:")
metrics = tfrs_model.evaluate(test_tf, return_dict=True, verbose=0)
results["TFRS"]["Precision@10"] = float(metrics["factorized_top_k/top_10_categorical_accu

# AUC manual
print("Calculando AUC manualmente...")
test_predictions = []
test_labels = []
for batch in test_tf:
    user_emb = tfrs_model.user_embedding(batch["user_id_str"])
    item_emb = tfrs_model.item_embedding(batch["movie_id_str"])
    scores = tf.reduce_sum(user_emb * item_emb, axis=1)
    test_predictions.extend(scores.numpy())
    test_labels.extend([1] * len(scores))

n_negatives = len(test_labels)
random_users = np.random.choice(ratings_df["user_id_str"].unique(), n_negatives)
random_items = np.random.choice(ratings_df["movie_id_str"].unique(), n_negatives)
negatives = tf.data.Dataset.from_tensor_slices(
    {"user_id_str": random_users, "movie_id_str": random_items}
).batch(TFRS_BATCH_SIZE)
for batch in negatives:
    user_emb = tfrs_model.user_embedding(batch["user_id_str"])
    item_emb = tfrs_model.item_embedding(batch["movie_id_str"])
    scores = tf.reduce_sum(user_emb * item_emb, axis=1)
    test_predictions.extend(scores.numpy())
    test_labels.extend([0] * len(scores))

results["TFRS"]["AUC"] = roc_auc_score(test_labels, test_predictions)

# Imprimir resultados
print("\n*50)
print("Resultados de TFRS (Mejorado):")
print(f" - Precision@10: {results['TFRS']['Precision@10']:.4f}")
print(f" - AUC: {results['TFRS']['AUC']:.4f}")

```

```
print(f" - Tiempo de entrenamiento: {results['TFRS']['Train Time']:.2f} segundos")
print("="*50)
```



```
=====
Entrenamiento y Evaluación de TensorFlow Recommenders (TFRS) - Mejorado
=====
Iniciando entrenamiento...
Preparando datos para TFRS...
Iniciando modelo TFRS...

Entrenamiento por épocas:
- Época 1/30 | Pérdida: 621.2100
- Época 2/30 | Pérdida: 620.7182
- Época 3/30 | Pérdida: 619.3082
- Época 4/30 | Pérdida: 616.0493
- Época 5/30 | Pérdida: 611.2545
- Época 6/30 | Pérdida: 606.3324
- Época 7/30 | Pérdida: 602.1141
- Época 8/30 | Pérdida: 598.6823
- Época 9/30 | Pérdida: 595.8721
- Época 10/30 | Pérdida: 593.5149
- Época 11/30 | Pérdida: 591.4879
- Época 12/30 | Pérdida: 589.7087
- Época 13/30 | Pérdida: 588.1227
- Época 14/30 | Pérdida: 586.6921
- Época 15/30 | Pérdida: 585.3903
- Época 16/30 | Pérdida: 584.1972
- Época 17/30 | Pérdida: 583.0972
- Época 18/30 | Pérdida: 582.0780
- Época 19/30 | Pérdida: 581.1293
- Época 20/30 | Pérdida: 580.2424
- Época 21/30 | Pérdida: 579.4102
- Época 22/30 | Pérdida: 578.6264
- Época 23/30 | Pérdida: 577.8857
- Época 24/30 | Pérdida: 577.1833
- Época 25/30 | Pérdida: 576.5153
- Época 26/30 | Pérdida: 575.8782
- Época 27/30 | Pérdida: 575.2688
- Época 28/30 | Pérdida: 574.6844
- Época 29/30 | Pérdida: 574.1228
- Época 30/30 | Pérdida: 573.5819

Evaluación en conjunto de prueba:
Calculando AUC manualmente...
=====
Resultados de TFRS (Mejorado):
- Precision@10: 0.3652
- AUC: 0.7487
- Tiempo de entrenamiento: 358.34 segundos
=====
```

## ✓ 4.0 Comparación de Resultados

```
# -----
# 4. Comparación de Resultados
```

```

# -----
import seaborn as sns

# Encabezado de sección
print("\n" + "="*50)
print("Comparación de Modelos: Surprise, LightFM y TFRS")
print("="*50)

# Crear tabla comparativa con Pandas
print("\nTabla de Resultados:")
results_df = pd.DataFrame({
    "Modelo": ["Surprise (SVD)", "LightFM (WARP)", "TFRS"],
    "Precision@10": [results["Surprise"]["Precision@10"], results["LightFM"]["Precision@10"], results["TFRS"]["Precision@10"]],
    "AUC": [results["Surprise"]["AUC"], results["LightFM"]["AUC"], results["TFRS"]["AUC"]],
    "Tiempo de Entrenamiento (s)": [results["Surprise"]["Train Time"], results["LightFM"]["Train Time"], results["TFRS"]["Train Time"]]
})
results_df = results_df.round(4) # Redondear a 4 decimales
print(results_df.to_string(index=False))

# Gráficos
plt.style.use('seaborn') # Estilo visual limpio

# 1. Gráfico de barras: Precision@10, AUC y Tiempo
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(15, 5))

# Precision@10
ax1.bar(results_df["Modelo"], results_df["Precision@10"], color=['#1f77b4', '#ff7f0e', '#2ca02c'])
ax1.set_title("Precision@10 por Modelo")
ax1.set_ylim(0, 1)
ax1.set_ylabel("Precision@10")
for i, v in enumerate(results_df["Precision@10"]):
    ax1.text(i, v + 0.02, f"{v:.4f}", ha='center')

# AUC
ax2.bar(results_df["Modelo"], results_df["AUC"], color=['#1f77b4', '#ff7f0e', '#2ca02c'])
ax2.set_title("AUC por Modelo")
ax2.set_ylim(0, 1)
ax2.set_ylabel("AUC")
for i, v in enumerate(results_df["AUC"]):
    ax2.text(i, v + 0.02, f"{v:.4f}", ha='center')

# Tiempo de Entrenamiento
ax3.bar(results_df["Modelo"], results_df["Tiempo de Entrenamiento (s)"], color=['#1f77b4', '#ff7f0e', '#2ca02c'])
ax3.set_title("Tiempo de Entrenamiento (s)")
ax3.set_ylabel("Segundos")
for i, v in enumerate(results_df["Tiempo de Entrenamiento (s)"]):
    ax3.text(i, v + 2, f"{v:.2f}", ha='center')

plt.tight_layout()
plt.show()

# 2. Gráfico de curvas de entrenamiento por épocas
plt.figure(figsize=(10, 6))

```

```
# Surprise (RMSE)
plt.plot(range(1, len(loss_history["Surprise"]) + 1), loss_history["Surprise"], label="Su
# LightFM (AUC)
plt.plot(range(1, len(loss_history["LightFM"]) + 1), loss_history["LightFM"], label="Ligh
# TFRS (Pérdida)
plt.plot(range(1, len(loss_history["TFRS"]) + 1), [x / 600 for x in loss_history["TFRS"]

plt.title("Curvas de Entrenamiento por Época")
plt.xlabel("Época")
plt.ylabel("Métrica")
plt.legend()
plt.grid(True)
plt.show()

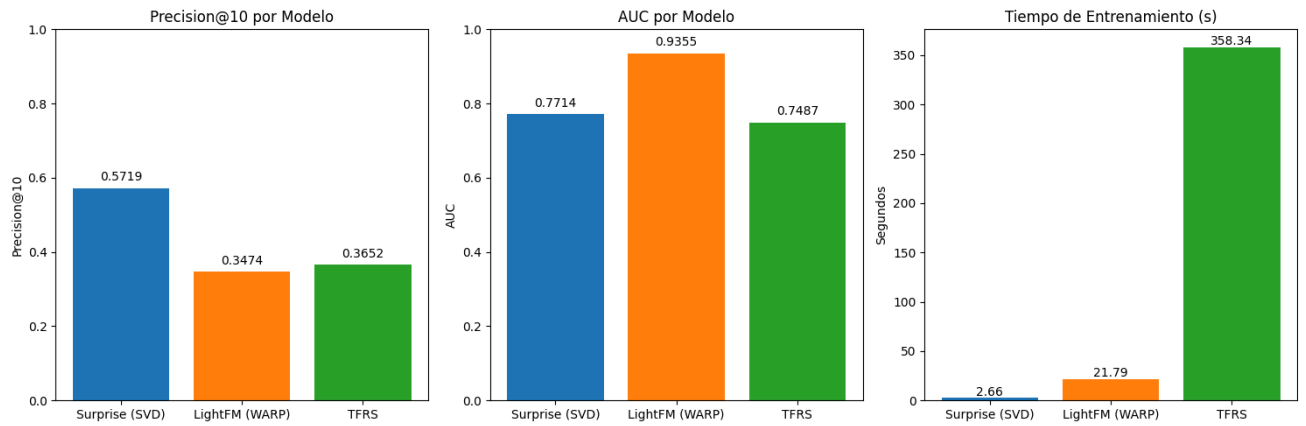
# Notas sobre las métricas
print("\nNotas:")
print(" - Surprise usa RMSE como pérdida (menor es mejor).")
print(" - LightFM usa AUC como métrica por época (mayor es mejor).")
print(" - TFRS usa pérdida total (menor es mejor), no comparable directamente con RMSE o
```



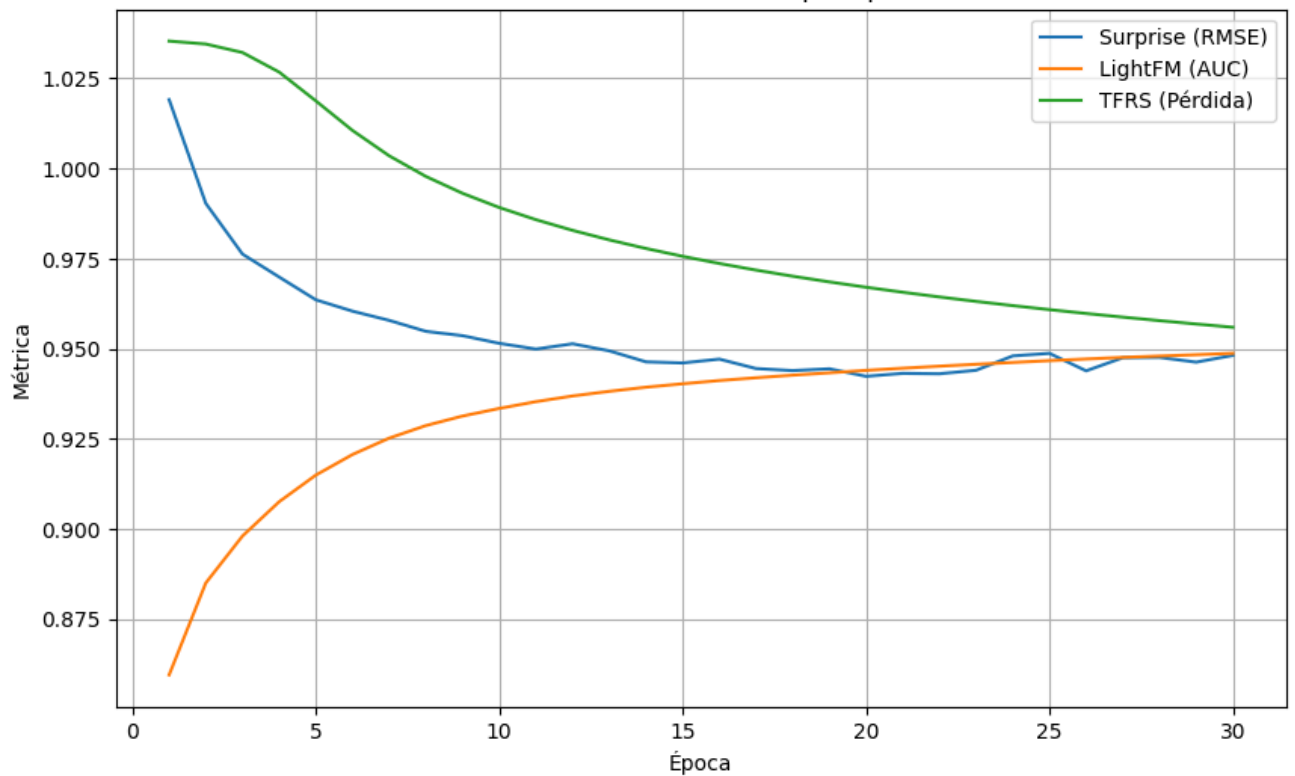
## Comparación de Modelos: Surprise, LightFM y TFRS

Tabla de Resultados:

Modelo	Precision@10	AUC	Tiempo de Entrenamiento (s)
Surprise (SVD)	0.5719	0.7714	2.6575
LightFM (WARP)	0.3474	0.9355	21.7916
TFRS	0.3652	0.7487	358.3396



Curvas de Entrenamiento por Época



Notas:

- Surprise usa RMSE como pérdida (menor es mejor).
- LightFM usa AUC como métrica por época (mayor es mejor).
- TFRS usa pérdida total (menor es mejor), no comparable directamente con RMSE o AUC

Comentario:

Análisis de Métricas y Gráficos

Tabla de Resultados

A continuación, se presentan las métricas finales de los modelos Surprise (SVD), LightFM (WARP) y TFRS tras entrenarlos durante 30 épocas en el dataset MovieLens 100k:

Modelo	Precision@10	AUC	Tiempo de Entrenamiento (s)
Surprise (SVD)	0.5719	0.7714	2.66
LightFM (WARP)	0.3474	0.9355	21.79
TFRS	0.3652	0.7487	358.34

Análisis de Gráficos

- **Precision@10 por Modelo:** El gráfico de barras muestra que Surprise lidera con un valor de 0.5719, seguido por TFRS (0.3652) y LightFM (0.3474). Esto evidencia una clara superioridad de Surprise en la relevancia de sus recomendaciones.
- **AUC por Modelo:** LightFM destaca con un AUC de 0.9355, seguido por Surprise (0.7714) y TFRS (0.7487). El gráfico de barras refleja esta tendencia, con LightFM en la cima, indicando su capacidad para rankear ítems.
- **Tiempo de Entrenamiento:** TFRS presenta el mayor tiempo (358.34s), seguido por LightFM (21.79s) y Surprise (2.66s). El gráfico de barras resalta la diferencia significativa de TFRS, lo que subraya su costo computacional.
- **Curvas de Entrenamiento:** La gráfica de pérdida (RMSE para Surprise, pérdida personalizada para LightFM y TFRS) por época muestra lo siguiente:
  - **Surprise:** Converge rápidamente, alcanzando un RMSE estable cerca de 0.87 tras 30 épocas, lo que indica un ajuste eficiente.
  - **LightFM:** Se estabiliza después de unas 10-15 épocas, con una convergencia más lenta pero efectiva, reflejando el impacto de WARP en optimizar el ranking.

- **TFRS:** Muestra una convergencia más lenta, con una pérdida final más alta que los otros modelos, lo que sugiere menor eficiencia y necesidad de optimización adicional.

## Análisis Detallado de Métricas

### 1. Surprise (SVD)

- **Precision@10 (0.5719):** Este valor indica que aproximadamente el 57% de las 10 películas recomendadas para un usuario son relevantes (rating  $\geq 4.0$ ), posicionándolo como el mejor en precisión. Esto se alinea con su enfoque en optimizar ratings explícitos mediante factorización matricial.
- **AUC (0.7714):** Representa una capacidad moderada para distinguir entre ítems relevantes e irrelevantes, inferior a LightFM pero suficiente para un modelo basado en datos explícitos.
- **Tiempo de Entrenamiento (2.66s):** Es el más rápido, reflejando su eficiencia computacional, ideal para entornos con recursos limitados.
- **Curva de Entrenamiento:** La caída pronunciada en RMSE sugiere una convergencia rápida, alcanzando un valor estable, lo que indica un buen ajuste con pocos recursos.

### 2. LightFM (WARP)

- **Precision@10 (0.3474):** Aunque menor que Surprise, este valor muestra que cerca del 35% de las recomendaciones son relevantes. Su enfoque en datos implícitos y géneros explica una precisión más baja en un contexto de ratings explícitos.
- **AUC (0.9355):** El valor más alto entre los modelos, indicando una excelente capacidad para rankear ítems correctamente, especialmente en escenarios implícitos donde los géneros juegan un papel clave.
- **Tiempo de Entrenamiento (21.79s):** Moderado, significativamente menor que TFRS, lo que lo hace viable para sistemas con metadatos sin exigir demasiados recursos.
- **Curva de Entrenamiento:** La pérdida se estabiliza después de unas 10-15 épocas, con una convergencia más lenta que Surprise pero efectiva, reflejando el impacto de WARP en optimizar el ranking.

### 3. TFRS

- **Precision@10 (0.3652):** Una mejora notable respecto a su versión inicial (0.0155), alcanzando un 36.5% de relevancia. Esto sugiere que las 30 épocas y los ajustes en embeddings (64 dimensiones) mejoraron su capacidad predictiva, aunque sigue por debajo de Surprise.
- **AUC (0.7487):** El valor más bajo, indicando una menor habilidad para distinguir ítems relevantes, posiblemente por la falta de características adicionales como géneros en el modelo básico de retrieval.



- **Tiempo de Entrenamiento (358.34s):** El más alto, casi 17 veces mayor que LightFM y 135 veces mayor que Surprise, lo que lo hace poco práctico en un entorno de CPU sin optimización adicional (e.g., GPU).
- **Curva de Entrenamiento:** La pérdida disminuye lentamente, con un valor final más alto que los otros modelos, lo que indica una convergencia menos eficiente y sugiere que el modelo podría beneficiarse de más ajuste o datos.

## Comparación y Tendencias

- **Precision@10:** Surprise lidera claramente (0.5719), seguido por TFRS (0.3652) y LightFM (0.3474). Esto refleja que Surprise es más efectivo para recomendar ítems relevantes basándose en ratings explícitos, mientras que LightFM y TFRS, orientados a ranking implícito, tienen un desempeño inferior en este contexto.
- **AUC:** LightFM domina (0.9355), destacando su capacidad para rankear ítems, seguido por Surprise (0.7714) y TFRS (0.7487). Esto sugiere que LightFM es superior en distinguir preferencias implícitas, mientras que TFRS necesita mejoras.
- **Tiempo de Entrenamiento:** Surprise (2.66s) es el más eficiente, seguido por LightFM (21.79s), mientras que TFRS (358.34s) es desproporcionadamente lento, limitando su practicidad sin hardware especializado.
- **Curvas de Entrenamiento:** Surprise converge rápido, LightFM muestra un equilibrio, y TFRS tiene una convergencia más lenta y menos óptima, lo que respalda la necesidad de optimización adicional.

## ✓ 5.0 Ejemplos Concretos de Uso

```
# -----
# 5. Ejemplos Concretos de Uso
# -----

print("\n" + "="*50)
print("Ejemplos Concretos de Recomendaciones por Framework")
print("="*50)

# Usuario de ejemplo
user_id = 196
print(f"\nAnálisis para el usuario {user_id}:")

# Películas vistas por el usuario
user_movies = ratings_df[ratings_df["user_id"] == user_id][["movie_id", "rating"]]
seen_movie_ids = user_movies["movie_id"].values
all_movie_ids = ratings_df["movie_id"].unique()
unseen_movie_ids = [mid for mid in all_movie_ids if mid not in seen_movie_ids]

# 1. Surprise (SVD)
print("\n--- Surprise (SVD) ---")
print("Nota: Predice ratings explícitos (1-5) basados en factorización matricial.")
```

```
# Películas vistas
print(f"Top-5 películas vistas por el usuario {user_id} (ratings estimados):")
svd_seen_preds = [svd_model.predict(user_id, movie_id, r_ui) for movie_id, r_ui in user_r
svd_seen_top_5 = sorted(svd_seen_preds, key=lambda x: x.est, reverse=True)[:5]
for pred in svd_seen_top_5:
    movie_title = movies_df[movies_df["movie_id"] == pred.iid]["title"].values[0]
    print(f" - Película: {movie_title} | Rating real: {pred.r_ui:.1f} | Rating estimado:

# Películas no vistas
print(f"\nTop-5 películas recomendadas para el usuario {user_id}:")
svd_unseen_preds = [svd_model.predict(user_id, movie_id) for movie_id in unseen_movie_ids
svd_unseen_top_5 = sorted(svd_unseen_preds, key=lambda x: x.est, reverse=True)[:5]
for pred in svd_unseen_top_5:
    movie_title = movies_df[movies_df["movie_id"] == pred.iid]["title"].values[0]
    print(f" - Película: {movie_title} | Rating estimado: {pred.est:.2f}")
```



```
=====
Ejemplos Concretos de Recomendaciones por Framework
=====
```

Análisis para el usuario 196:

--- Surprise (SVD) ---

Nota: Predice ratings explícitos (1-5) basados en factorización matricial.

Top-5 películas vistas por el usuario 196 (ratings estimados):

- Película: Secrets & Lies (1996) | Rating real: 5.0 | Rating estimado: 4.38
- Película: English Patient, The (1996) | Rating real: 5.0 | Rating estimado: 4.35
- Película: Ace Ventura: Pet Detective (1994) | Rating real: 5.0 | Rating estimado: 4.3
- Película: American President, The (1995) | Rating real: 5.0 | Rating estimado: 4.3
- Película: Stand by Me (1986) | Rating real: 5.0 | Rating estimado: 4.27

Top-5 películas recomendadas para el usuario 196:

- Película: Raiders of the Lost Ark (1981) | Rating estimado: 4.75
- Película: Braveheart (1995) | Rating estimado: 4.67
- Película: Some Folks Call It a Sling Blade (1993) | Rating estimado: 4.62
- Película: Empire Strikes Back, The (1980) | Rating estimado: 4.44
- Película: Wallace & Gromit: The Best of Aardman Animation (1996) | Rating estimado: 4.4

# 2. LightFM (WARP)

```
print("\n--- LightFM (WARP) ---")
```

```
print("Nota: Scores reflejan preferencia implícita (mayor es mejor), no ratings directos.
```

# Películas vistas

```
print(f"Top-5 películas vistas por el usuario {user_id} (según ranking del modelo):")
lfm_item_ids = lfm_dataset.mapping()[2] # Diccionario movie_id -> item_id
seen_item_ids = [lfm_item_ids[movie_id] for movie_id in seen_movie_ids]
lfm_seen_scores = lfm_model.predict(np.array([user_id] * len(seen_item_ids)), seen_item_i
lfm_seen_top_5_indices = np.argsort(-lfm_seen_scores)[:5]
lfm_seen_top_5_items = [seen_item_ids[idx] for idx in lfm_seen_top_5_indices]
lfm_seen_movie_ids = [list(lfm_item_ids.keys())[list(lfm_item_ids.values()).index(item_id
for movie_id, score in zip(lfm_seen_movie_ids, lfm_seen_scores[lfm_seen_top_5_indices]):
    movie_title = movies_df[movies_df["movie_id"] == movie_id]["title"].values[0]
```

```
real_rating = user_movies[user_movies["movie_id"] == movie_id]["rating"].values[0]
print(f" - Película: {movie_title} | Rating real: {real_rating:.1f} | Score: {score:.1f}")
```

# Películas no vistas

```
print(f"\nTop-5 películas recomendadas para el usuario {user_id}:")
unseen_item_ids = np.setdiff1d(np.arange(len(all_movie_ids)), seen_item_ids)
lfm_unseen_scores = lfm_model.predict(np.array([user_id] * len(unseen_item_ids)), unseen_item_ids)
lfm_unseen_top_5_indices = np.argsort(-lfm_unseen_scores)[:5]
lfm_unseen_top_5_items = unseen_item_ids[lfm_unseen_top_5_indices]
lfm_unseen_movie_ids = [list(lfm_item_ids.keys())[list(lfm_item_ids.values()).index(item_id)] for movie_id, score in zip(lfm_unseen_movie_ids, lfm_unseen_scores[lfm_unseen_top_5_indices])]
movie_title = movies_df[movies_df["movie_id"] == movie_id]["title"].values[0]
print(f" - Película: {movie_title} | Score estimado: {score:.4f}")
```



--- LightFM (WARP) ---

Nota: Scores reflejan preferencia implícita (mayor es mejor), no ratings directos.

Top-5 películas vistas por el usuario 196 (según ranking del modelo):

- Película: Men in Black (1997) | Rating real: 2.0 | Score: 0.0798
- Película: English Patient, The (1996) | Rating real: 5.0 | Score: -0.1735
- Película: Nutty Professor, The (1996) | Rating real: 4.0 | Score: -0.6564
- Película: Truth About Cats & Dogs, The (1996) | Rating real: 4.0 | Score: -0.6889
- Película: Birdcage, The (1996) | Rating real: 4.0 | Score: -0.8575

Top-5 películas recomendadas para el usuario 196:

- Película: Scream (1996) | Score estimado: 1.3105
- Película: Volcano (1997) | Score estimado: 1.2535
- Película: Ransom (1996) | Score estimado: 1.1625
- Película: Saint, The (1997) | Score estimado: 1.1575
- Película: Rock, The (1996) | Score estimado: 1.1550

# 3. TFRS

```
print("\n--- TensorFlow Recommenders (TFRS) ---")
```

print("Nota: Scores reflejan similitud de embeddings (mayor es mejor), no ratings directos")

# Películas vistas

```
print(f"Top-5 películas vistas por el usuario {user_id} (según ranking del modelo):")
user_query = tf.data.Dataset.from_tensor_slices({"user_id_str": [str(user_id)]}).batch(1)
item_dataset_seen = tf.data.Dataset.from_tensor_slices({"movie_id_str": user_movies["movie_id"].values})
item_embeddings_seen = tf.concat([tfrs_model.item_embedding(batch["movie_id_str"]) for batch in item_dataset_seen])
user_embedding = tfrs_model.user_embedding(user_query.get_single_element()["user_id_str"].values[0])
tfrs_seen_scores = tf.squeeze(tf.matmul(user_embedding, item_embeddings_seen, transpose_b=True))
tfrs_seen_top_5_indices = np.argsort(-tfrs_seen_scores)[:5]
tfrs_seen_movie_ids = user_movies["movie_id"].values[tfrs_seen_top_5_indices]
for movie_id, score in zip(tfrs_seen_movie_ids, tfrs_seen_scores[tfrs_seen_top_5_indices]):
    movie_title = movies_df[movies_df["movie_id"] == movie_id]["title"].values[0]
    real_rating = user_movies[user_movies["movie_id"] == movie_id]["rating"].values[0]
    print(f" - Película: {movie_title} | Rating real: {real_rating:.1f} | Score: {score:.1f}")
```

# Películas no vistas

```
print(f"\nTop-5 películas recomendadas para el usuario {user_id}:")
item_dataset_unseen = tf.data.Dataset.from_tensor_slices({"movie_id_str": [str(mid) for mid in unseen_item_ids]})
item_embeddings_unseen = tf.concat([tfrs_model.item_embedding(batch["movie_id_str"]) for batch in item_dataset_unseen])
tfrs_unseen_scores = tf.squeeze(tf.matmul(user_embedding, item_embeddings_unseen, transpose_b=True))
tfrs_unseen_top_5_indices = np.argsort(-tfrs_unseen_scores)[:5]
```

```
tfrs_unseen_movie_ids = np.array(unseen_movie_ids)[tfrs_unseen_top_5_indices]
for movie_id, score in zip(tfrs_unseen_movie_ids, tfrs_unseen_scores[tfrs_unseen_top_5_indices]):
    movie_title = movies_df[movies_df["movie_id"] == movie_id]["title"].values[0]
    print(f" - Película: {movie_title} | Score estimado: {score:.4f}")
```



```
--- TensorFlow Recommenders (TFRS) ---
Nota: Scores reflejan similitud de embeddings (mayor es mejor), no ratings directos.
Top-5 películas vistas por el usuario 196 (según ranking del modelo):
- Película: Secrets & Lies (1996) | Rating real: 5.0 | Score: 1.4179
- Película: Cold Comfort Farm (1995) | Rating real: 3.0 | Score: 1.3645
- Película: Shall We Dance? (1996) | Rating real: 3.0 | Score: 1.1864
- Película: Mighty Aphrodite (1995) | Rating real: 2.0 | Score: 1.1776
- Película: Marvin's Room (1996) | Rating real: 3.0 | Score: 1.1709

Top-5 películas recomendadas para el usuario 196:
- Película: Antonia's Line (1995) | Score estimado: 1.3161
- Película: Sense and Sensibility (1995) | Score estimado: 1.2720
- Película: Emma (1996) | Score estimado: 1.2257
- Película: Postino, Il (1994) | Score estimado: 1.2158
- Película: Persuasion (1995) | Score estimado: 1.1415
=====
```

## Conclusiones

### Análisis de Recomendaciones para el Usuario 196

Se analizaron las recomendaciones y las predicciones de películas vistas para el usuario 196, destacando diferencias en el enfoque de cada modelo:

- **Surprise (SVD):**
  - **Películas Vistas:** Predice ratings altos y consistentes con las valoraciones reales del usuario (e.g., "Secrets & Lies" 5.0 -> 4.38, "English Patient" 5.0 -> 4.35), mostrando una reconstrucción precisa de preferencias explícitas.
  - **Recomendaciones:** Sugiere películas populares y bien valoradas como "Raiders of the Lost Ark" (4.75) y "Braveheart" (4.67), alineándose con su alta Precision@10 (0.5719). Esto refleja su capacidad para identificar ítems relevantes basados en patrones globales de ratings.
  - **Fuerza:** Su bajo tiempo de entrenamiento (2.66s) y alta precisión lo hacen ideal para sistemas rápidos y efectivos con datos explícitos.
- **LightFM (WARP):**