From Transformers to Preferences: Bridging LLMs and Recommender Systems

Alessandro Petruzzelli

PhD Student, University of Bari

Today's Journey

1. **Part 1: Foundations: What is the Recommendation Problem?**
   - *From Static Matrices to Sequential Transformers*
2. **Part 2: The LLM-RecSys Playbook**
   - *From Encoders to Generators*
3. **Part 3: Evaluation, Risks, and Grand Challenges**
   - *Evaluating the "Un-evaluable" & The Future*
4. **Part 4: Practical Lab: Building a Transformer-based Recommender**
   - *Hands-on with Bert4Rec (You already know how to do this!)*

Part 1: Foundations

What is the Recommendation Problem?

The "Problem" is Everywhere

- **Spotify:** "What song should you listen to next?"
- **Netflix:** "What movie should you watch tonight?"
- **Amazon:** "What product should you buy with this?"

At its core, a Recommender System (RecSys) is a tool for **taming information overload**.

Its job is to find a *tiny* set of relevant items from a *massive* catalog (billions of items) and predict a user's **preference**.

Module 1: The Classic Problem

We start with the **User-Item Interaction Matrix**, R.

This is the foundational data structure for **ID-Based Recommendations**.

- **Rows:** All your users (e.g., $U_1, U_2, \ldots, U_m$)
- **Columns:** All your items (e.g., $I_1, I_2, \ldots, I_n$)
- **Cells** $R_{u,i}$: A value representing preference.
    - **Explicit Feedback:** A user's *rating* (e.g., 1-5 stars).
    - **Implicit Feedback:** A user's *action* (e.g., 1=clicked, 0=not clicked).

Our goal: The matrix is 99.9% empty. Our job is to fill in the blanks.

The Core Challenge: Sparsity

Most users have only interacted with a tiny fraction of items.

The question: How do we predict the preference for R_{u_2,i_3}?

Classic Approaches (Briefly)

1. **Content-Based Filtering:**
    - "You liked *this item*, so you'll like *items with similar features*."
    - **Analogy:** k-Nearest Neighbors in the *item feature space*.
    - **Problem:** Creates a "filter bubble." Low novelty.
2. **Collaborative Filtering (CF):**
    - "You are *similar to other users*. Therefore, you will like *items they liked*."
    - **Analogy:** k-Nearest Neighbors in the *user vector space*.
    - **Problem:** Fails for new users/items (the "cold start" problem).

Classic Approach: Matrix Factorization

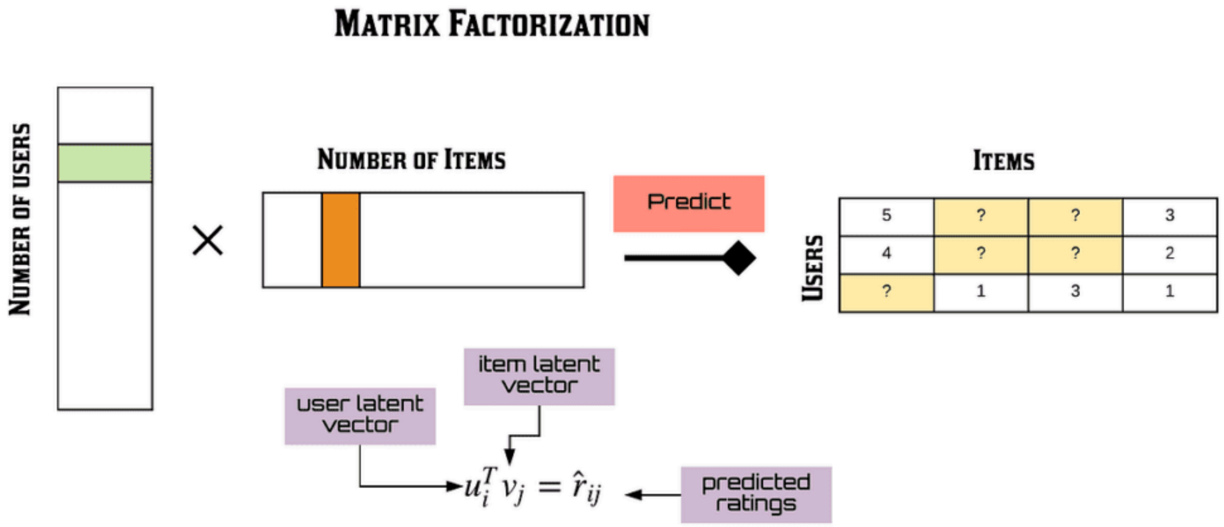
Instead of k-NN, let's learn a dense, low-dimensional *latent representation* (an **embedding**) for every user and every item.

- We learn a **User-Factor** vector $\gamma_u \in \mathbb{R}^k$
- We learn an **Item-Factor** vector $\gamma_i \in \mathbb{R}^k$

The predicted rating $\hat{r}_{u,i}$ is simply the dot product: $\hat{r}_{u,i} \approx \gamma_u \cdot \gamma_i$.

Matrix Factorization

We "solve" the sparse matrix R by approximating it as the product of two dense, low-rank ("thin") matrices, U and V.



Bridge 1: It's All Embeddings!

You already know this concept as **word embeddings** (like word2vec).

- **word2vec:** Learns vectors from word co-occurrence in a sentence.
- **Matrix Factorization:** Learns vectors from **user-item co-occurrence** in a preference matrix.

The goal is the same: find a vector γ where similarity (e.g., dot product) represents a meaningful relationship.

The Limit of the Static Matrix

The matrix view is **static**. It assumes your preferences are fixed.

But user preferences are dynamic:

- You buy a laptop. Your *next* action is to look for a laptop case.
- You watch a 2-hour action movie. Your *next* action is probably *not* another 2-hour action movie.

The *order* and *context* of your interactions matter.
The real problem isn't "what items do you like?" it's...

"What item do you want *right now*?"

Module 2: The Sequential Revolution

This moves us from a static problem to a **dynamic, sequential** one.

This is the *true* bridge to modern LLMs.

The New Problem: Next-Item Prediction

We discard the static matrix. Our data is now a **sequence of interactions**.

- **User History:** $S_u = (i_{u,1}, i_{u,2}, \ldots, i_{u,t})$
- **Session:** $(i_1, i_2, \ldots, i_t)$

The New Task: Given the user's history, predict the *next item* i_{t+1} they will interact with.

Approach 1: Markov Chains (MC)

The simplest sequential model.

- **Assumption:** The probability of the next item i_{t+1} depends *only* on the *current* item i_t
- **Formalization:** $p(i_{t+1}|i_1, \ldots, i_t) = p(i_{t+1}|i_t)$
- **Analogy:** This is a **bigram (n=2) model** from classic NLP.
- **Limitation:** "A model with the memory of a goldfish." It has no concept of "iPhone 15" when recommending "AirPods Pro" two steps later.

The Deep Learning Solutions

If a user's history is a "sentence," we can use the same models NLP uses to "read" it.

1. **RNNs** $\rightarrow$ **GRU4Rec** (2015)
2. **Decoder-Only Transformers** $\rightarrow$ **SASRec** (2018)
3. **Encoder-Only Transformers** $\rightarrow$ **BERT4Rec** (2019)

Let's look at the blocks.

Model 1: GRU4Rec (The "RNN" of RecSys)[1.]

- **Concept:** Use a Gated Recurrent Unit (GRU) to "read" the sequence of item embeddings and build a "session state."
- **How it works:**
    i. The user clicks item i_t.
    ii. We look up its embedding e_{i_t}.
    iii. We feed this into the GRU with the *previous* hidden state h_{t-1}.
    iv. The new hidden state h_t becomes our "session embedding," summarizing everything seen so far.
    v. This h_t is used to predict the *next* item i_{t+1}.

GRU4Rec: Formalization

1. **Input:** At step t, the one-hot vector for item i_t is embedded:
   $e_{i_t} = E \cdot i_t$ (where E is the item embedding matrix)

2. **Recurrence:** The hidden state is updated:
   $h_t = \mathrm{GRU}(e_{i_t}, h_{t-1})$

3. **Prediction:** The *score* for every *other* item j in the vocabulary V is calculated from this hidden state. A common way is a dot product:
   $\mathrm{score}(i_j, t) = h_t^T \cdot e_j$

4. **Loss:** Train with a session-parallel, mini-batch **Cross-Entropy Loss** (or BPR) on the *next* item i_{t+1}.

The Problem with RNNs

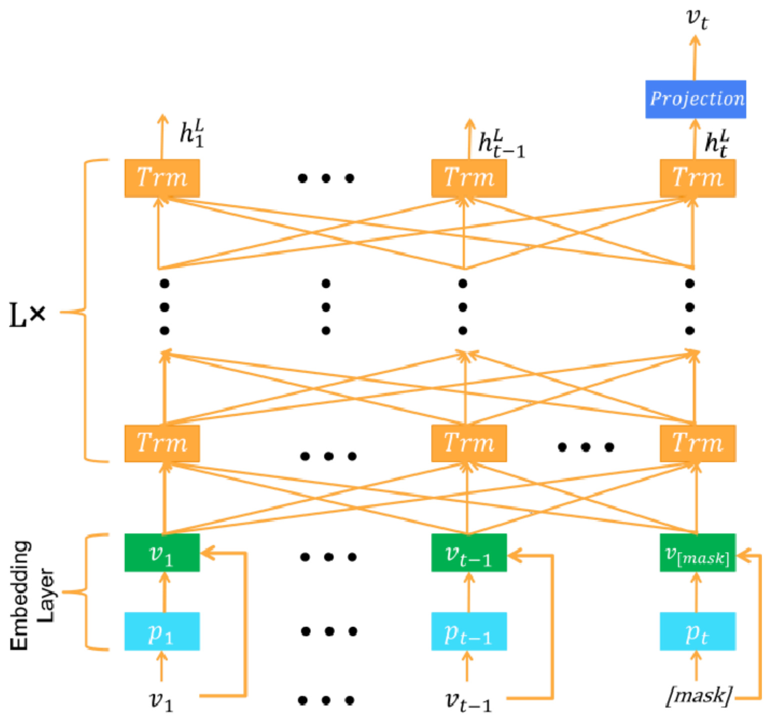
Why did NLP largely abandon RNNs for Transformers?

1. **Sequential Computation:** Cannot be parallelized over the time dimension. Training is *slow*.
2. **Vanishing/Exploding Gradients:** While GRUs/LSTMs help, they still struggle to model very long-range dependencies (e.g., an item you clicked 100 steps ago).

Recommender systems faced the *exact same problem*.
So, they adopted the *exact same solution*.

Model 2: BERT4Rec[3.]

- **Concept:** Use a bidirectional **Encoder-Only** Transformer.
- **Analogy:** This is the **"BERT"** of RecSys.
- **The Problem:** You can't use bidirectional attention for *next-item* prediction. The model could "see the future" and know the answer.
- **The Solution:** We invent a new task, just like BERT did.



BERT4Rec: Masked Item Prediction

Instead of Next-Item Prediction, we use **Masked Item Prediction**.

1. **Take a history:** `[i_1, i_2, i_3, i_4, i_5, i_6]`
2. **Randomly mask 15%:** `[i_1, i_2, [M], i_4, [M], i_6]`
3. **Train:** Use the full, bidirectional context to predict the *original* items `i_3` and `i_5`.

BERT4Rec: Architecture & Loss

1. **Input Embedding:**
   - $E_{input} = \text{ItemEmb}(I) + \text{PosEmb}(P)$
   - The `[M]` token is a special, learned `[MASK]` embedding.

BERT4Rec: Architecture & Loss

1. **Input Embedding**
2. **Transformer Blocks:**
    - The sequence is fed through L layers of standard Transformer **Encoder** blocks (bidirectional self-attention).
    - $H = \text{TransformerEncoder}(E_{input})$

BERT4Rec: Architecture & Loss

1. **Input Embedding**
2. **Transformer Blocks**
3. **Prediction:**
   - Take the hidden states $h_{[M]}$ corresponding to the *masked* positions.
   - Project them to the vocabulary: $\hat{y}_{[M]} = \mathrm{Softmax}(W_o h_{[M]} + b_o)$

BERT4Rec: Architecture & Loss

1. **Input Embedding**
2. **Transformer Blocks**
3. **Prediction**
4. **Loss:**
   - **Cross-Entropy Loss**, but *only* on the masked positions.

BERT4Rec: The "Inference Hack"

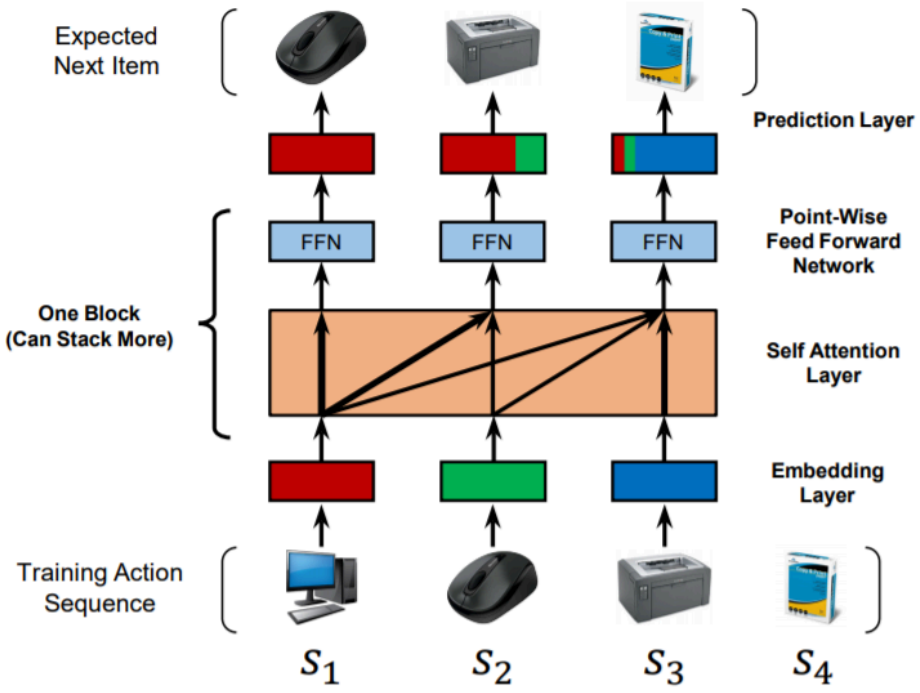
Wait... if it's trained to fill in the *middle*, how do we predict the *end*?

This is the clever (and slightly weird) part:

1. Take the user's *actual* history: `[i_1, i_2, i_3, i_4, i_5]`
2. **Append a `[MASK]` token** to the very end: `[i_1, i_2, i_3, i_4, i_5, [M]]`
3. Feed this *new sequence* into the trained BERT4Rec model.
4. The model's prediction for that *final* `[M]` token is our **next-item recommendation**.

Model 3: SASRec[2]

- **Concept: S**elf-**A**ttentive **S**equential **Rec**ommendation.
- **Analogy:** This is the **"GPT"** of RecSys.
- **Why?**
  i. It's an **Autoregressive** model.
  ii. It uses **Decoder-Only** Transformer blocks.
  iii. It's trained on **Next-Item Prediction**.
  iv. It uses **Causal (Look-Ahead) Masking**.



SASRec: Architecture & Formalization

1. **Input Embedding:** This is *identical* to GPT.
    - An **Item Embedding** (like a token embedding).
    - A learned **Positional Embedding** (to know the order).
    - $E_{input} = \mathrm{ItemEmb}(I) + \mathrm{PosEmb}(P)$

SASRec: Architecture & Formalization

1. **Input Embedding**
2. **Transformer Blocks:** The embedded sequence is fed through L layers of standard Transformer **Decoder** blocks.
    - $S_l = \text{CausalSelfAttention}(E_{input})$
    - $S_{l+1} = \text{FFN}(S_l)$

SASRec: Architecture & Formalization

1. **Input Embedding**
2. **Transformer Blocks**
3. **Causal Mask:** The self-attention is masked. The prediction for item at $t = 3$ can *only* see items at $t = 1, 2$.

SASRec: Prediction & Loss

This is also just like GPT.

1. **Prediction:**
    - We take the *final hidden state* s_t from the last Transformer block (corresponding to the *last item* i_t).
    - We calculate its dot product against *all item embeddings* E.
    - $\text{score}(i_j) = s_t^T \cdot e_j$ (for all j in the catalog)

SASRec: Prediction & Loss

This is also just like GPT.

1. **Prediction**
2. **Loss:**
   - We use a standard **Cross-Entropy Loss**. We want to maximize the score of the *true* next item i_{t+1}.
   - $L = -\log\left(\frac{\exp(\text{score}(i_{t+1}))}{\sum_{j \in V} \exp(\text{score}(i_j))}\right)$

SASRec: Prediction & Loss

This is also just like GPT.

1. **Prediction**
2. **Loss**
3. **Inference:**
    - `reco_list = torch.topk(scores, 10)`

Part 2: The LLM-RecSys Playbook

Architectures & State-of-the-Art Research

Module 3: The New "Data"

Unifying Modalities with Text

The Flaw in ID-Based Models

Models like `SASRec` and `BERT4Rec` are powerful, but they learn from **Item IDs** only.

1. **The Cold-Start Problem:**
   - If a new item `item_99999` appears, the model has **no embedding** for it.
   - The model is useless for new items until it's retrained.
2. **They are Data Hungry:**
   - The model has to learn the relationship between `item_732` (Inception) and `item_101` (The Matrix) *from scratch*, based only on user co-interactions.
   - It has no "world knowledge" that they are both "Sci-Fi thrillers."

The LLM Shift: From IDs to Text

Why not use a model that **already understands the world?**

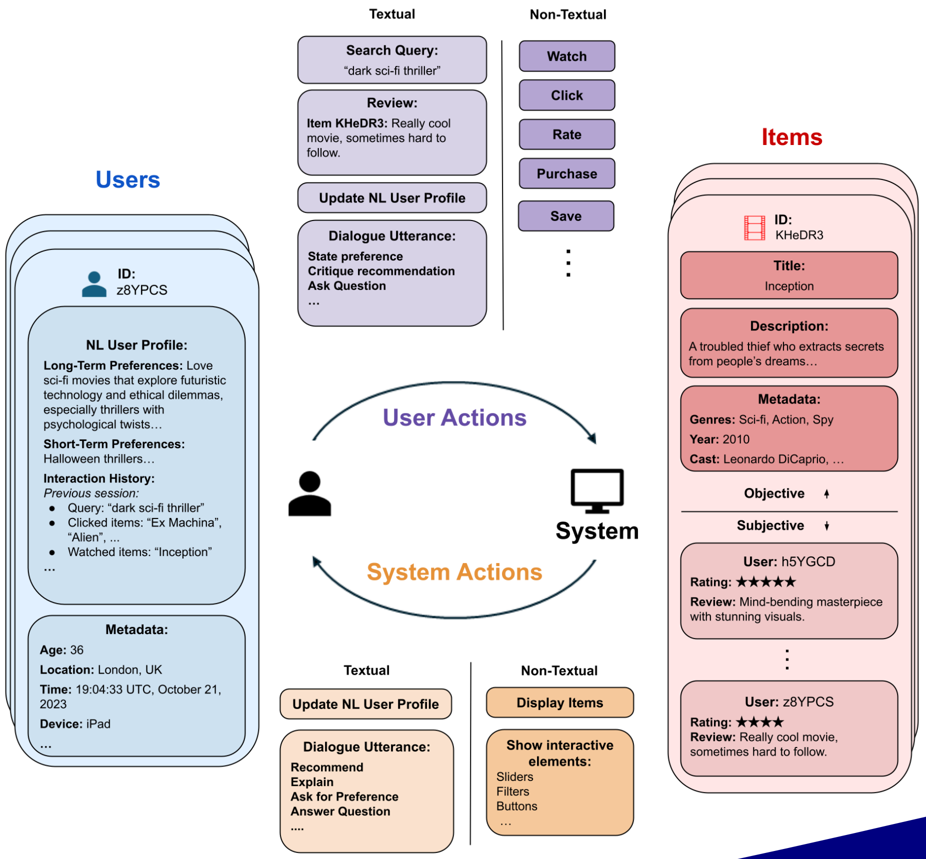
- A pre-trained LLM already knows the relationship between "Inception" and "The Matrix" from reading Wikipedia, reviews, and blogs.
- We shift from using abstract `item_ID` as the input to using the item's **text** (title, description, reviews.

This **solves the new-item cold-start problem**. We can create a meaningful embedding for a new item *immediately* just by encoding its description.

"Verbalization": The Unification of Data

Text becomes the **universal interface**.

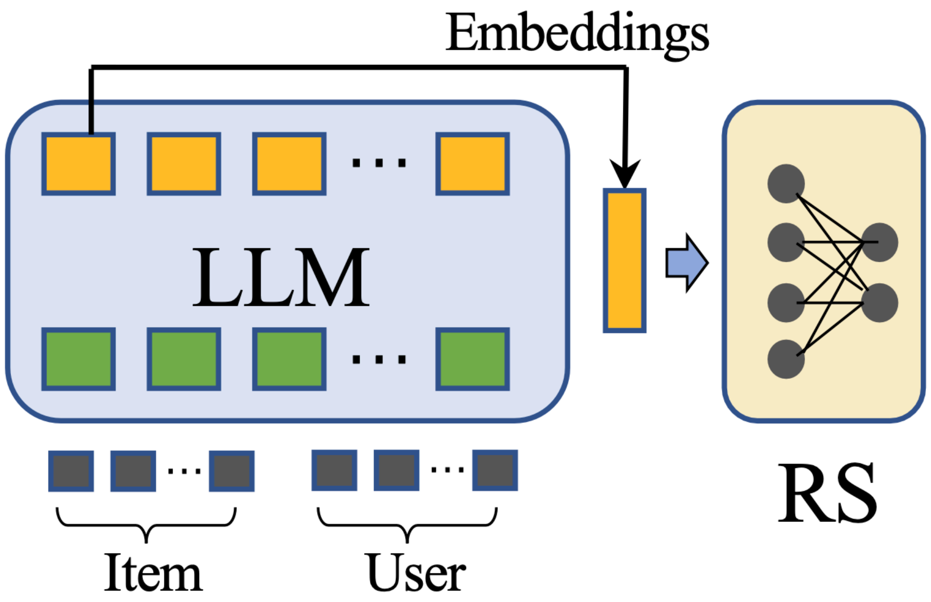
- **Item Data (Right):** Titles, Descriptions, Metadata, Reviews.
- **User Data (Left):** NL Profiles, Interaction History.
- **Interaction Data (Center):** *Textual* and *Non-Textual*.



The Three Modeling Paradigms

LLM-RecSys can be categorized into three main paradigms.

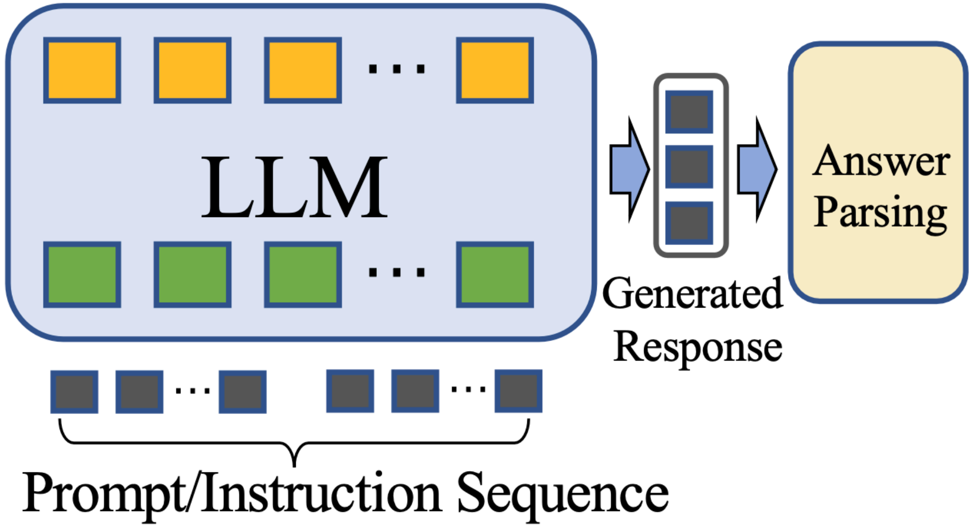
1. **LLM Embeddings + RS (LLM as Encoder)**
    - The LLM's job is to be a **feature extractor**, outputting high-quality embeddings.
    - A *separate, downstream* `RS` model makes the final prediction.



The Three Modeling Paradigms

LLM-RecSys can be categorized into three main paradigms.

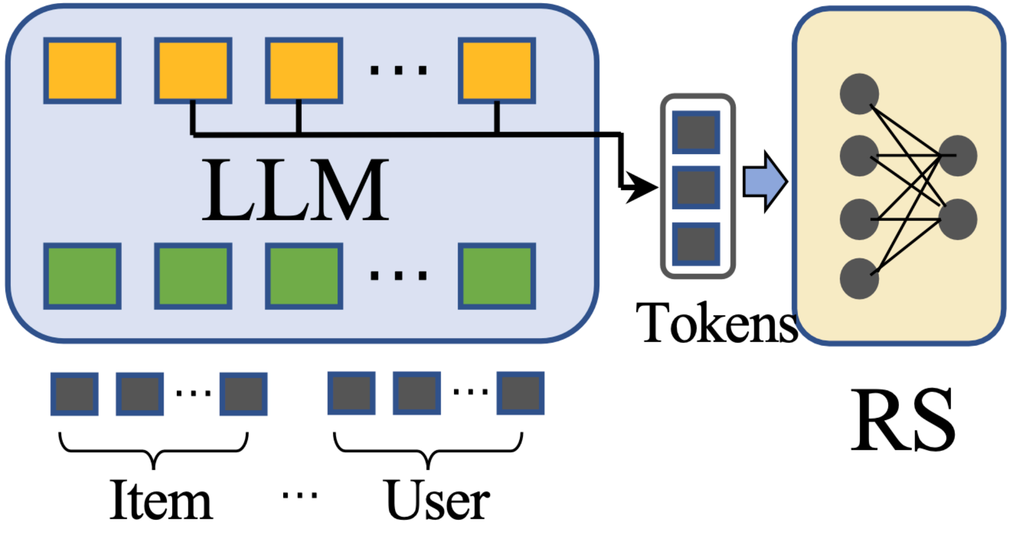
2. **LLM as RS (LLM as Generator)**
    - The LLM *is* the recommender.
    - It takes a prompt and **generates the final answer** (e.g., "The Three-Body Problem") directly.



The Three Modeling Paradigms

LLM-RecSys can be categorized into three main paradigms.

3. **LLM Tokens + RS (LLM as Conductor)**
    - The LLM is a "brain" in a multi-step process.
    - It **generates tokens** (e.g., preference summaries) that *another* component (like an `RS` or `Agent`) uses.

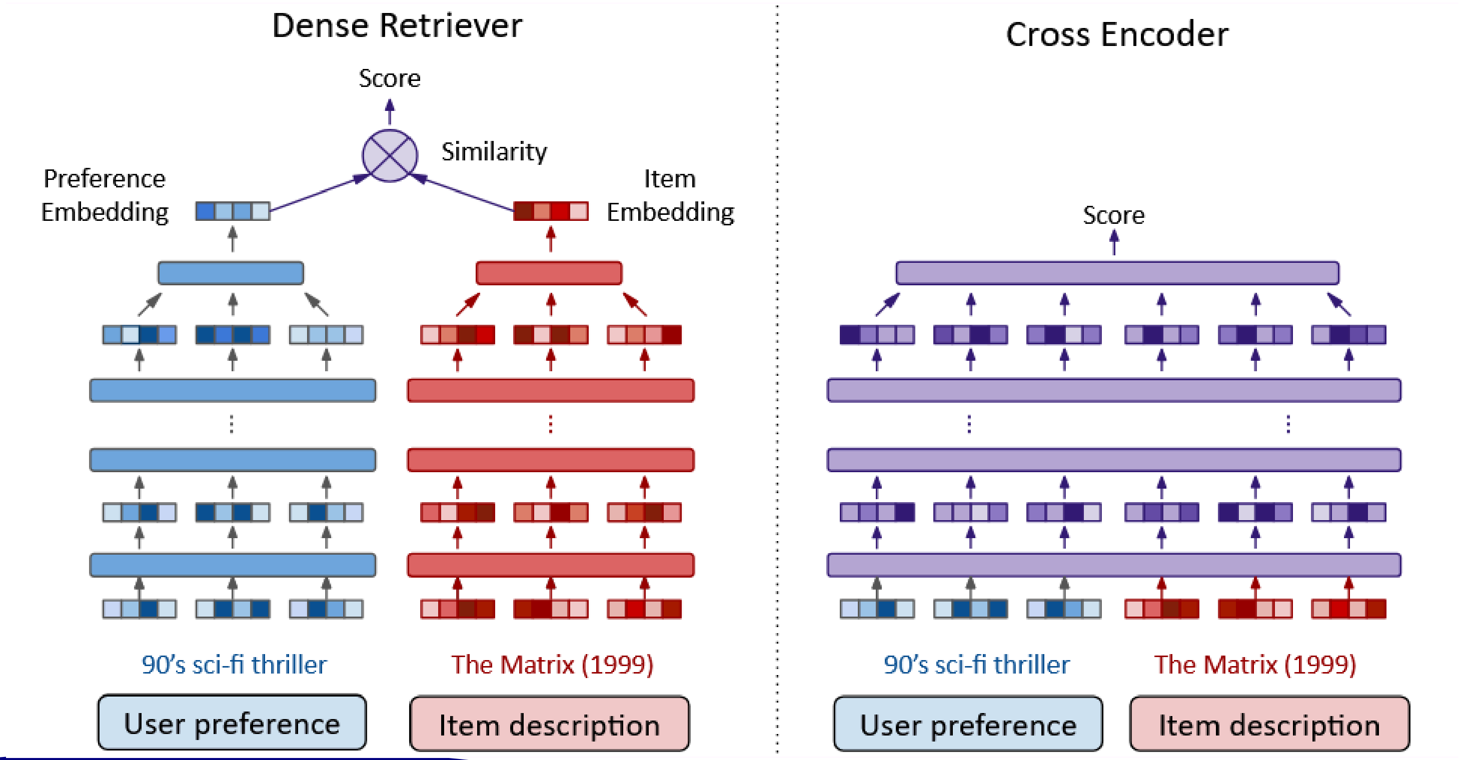


Module 4: Paradigm 1

`DLLM4Rec` (LLMs as Feature Encoders)

DLLM4Rec : Core Architectures

This paradigm uses an **Encoder-Only** model (like BERT) as a feature extractor.



1. **Dense Retrievers (Two-Tower):**
   - Encode user preference and item text *separately*, then compute similarity.
   - **Pro:** Extremely fast at inference. **Con:** Shallow interaction.
2. **Cross-Encoders (Re-ranking):**
   - *Concatenate* user and item text, then encode *jointly*.
   - **Pro:** High accuracy (deep interaction). **Con:** Extremely slow.

`DLLM4Rec`: Training Methods

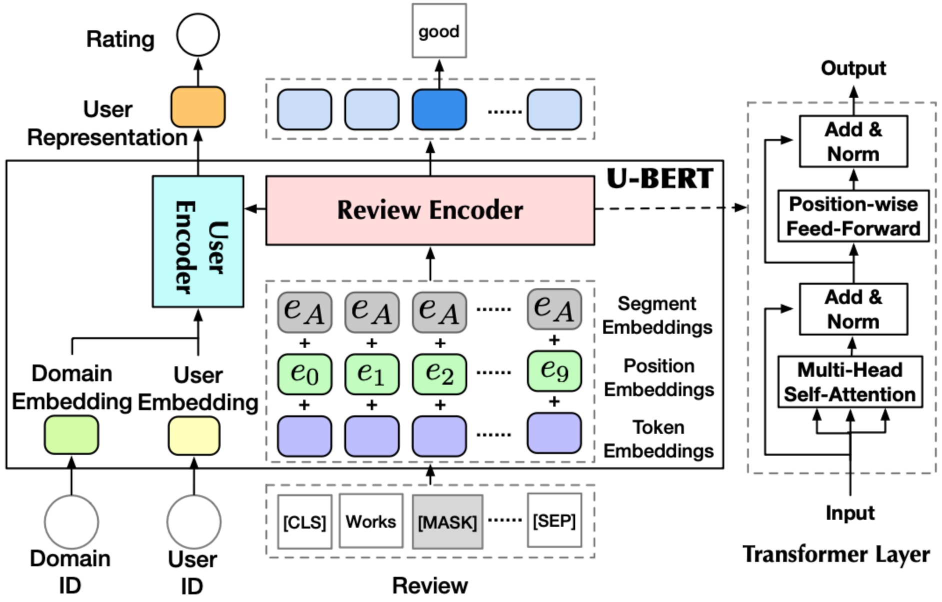
We can train these DLLM encoders in two primary ways:

1. **Fine-tuning:** The DLLM (e.g., BERT) is trained end-to-end with the RS model. Its weights are *updated* to optimize a specific `RS` loss (e.g., predict rating or click.)
2. **Prompt-Tuning:** The DLLM is framed as a **Masked Language Model (MLM)**. We design a "prompt" and train the model to predict a verbalized label (e.g., "good" or "bad") for the `[MASK]` token.

`DLLM4Rec` U-BERT [4]

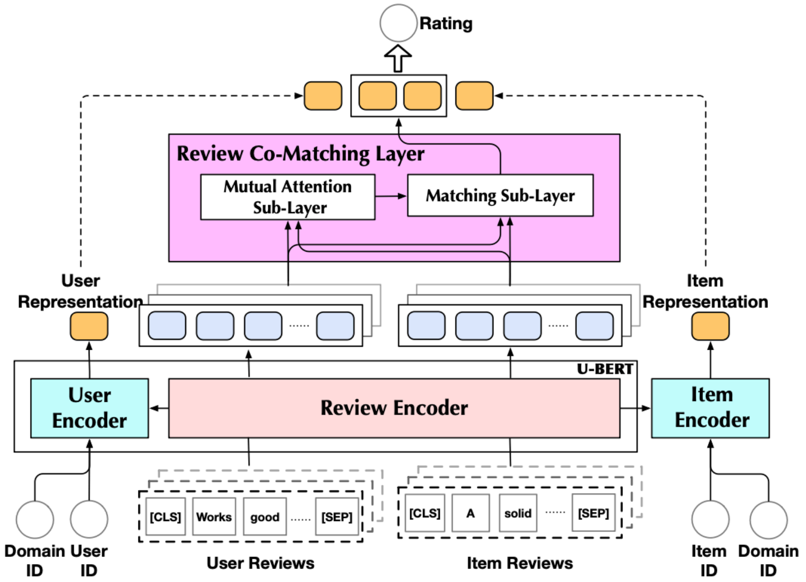
U-BERT is a classic example of the Fine-Tuning paradigm.

- **Stage 1: Pre-training**
  - The model is pre-trained on a "masked word" prediction task, using domain/user IDs as segment embeddings. This teaches the model the *language* of reviews.



`DLLM4Rec` U-BERT

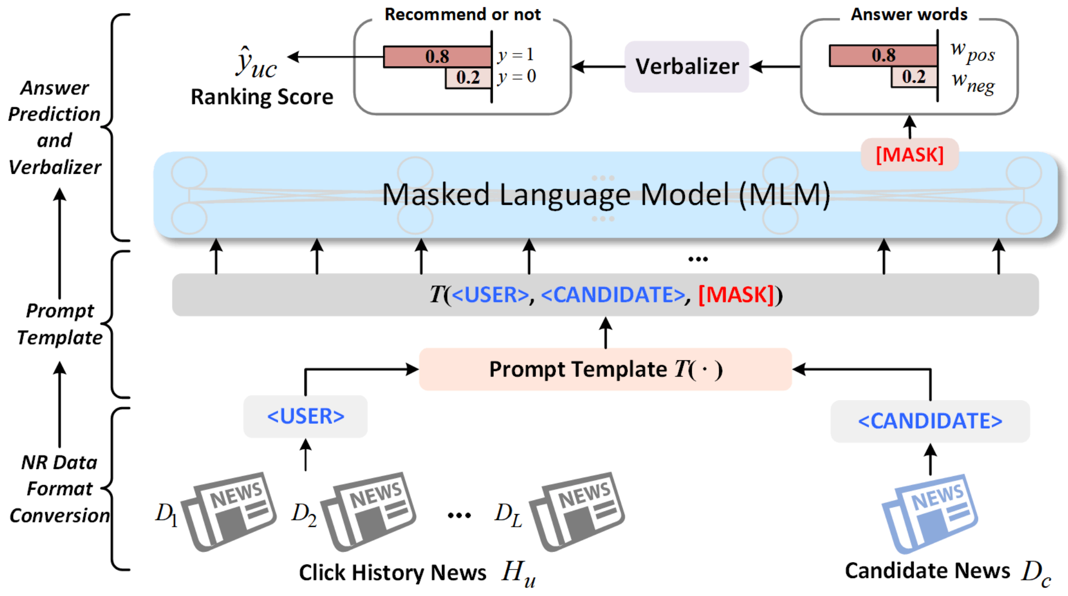
- **Stage 2: Fine-tuning**
  - The pre-trained U-BERT is then used as an encoder in a *downstream task*.
  - It encodes the User's Reviews and the Item's Reviews, and a "Co-Matching Layer" predicts the final rating.



`DLLM4Rec` Prompt4NR [5]

Prompt4NR is a perfect example of prompt-tuning.

- **Concept:** Reformulates news recommendation as a "fill-in-the-blank" task.



`DLLM4Rec` Prompt4NR

Types	Perspectives	Templates T(<USER>, <CANDIDATE>, [MASK])	Answer Words
Discrete Template	Relevance	<CANDIDATE> is [MASK] to <USER>	related/unrelated
	Emotion	The user feels [MASK] about <CANDIDATE> according to his area of interest <USER>	interesting/boring
	Action	User: <USER> [SEP] News: <CANDIDATE> [SEP] Does the user click the news? [MASK]	yes/no
	Utility	Recommending <CANDIDATE> to the user is a [MASK] choice according to <USER>	good/bad
Continuous Template	Relevance	$[Q_1]...[Q_{n_2}]$ <CANDIDATE> $[M_1]...[M_{n_3}]$ [MASK] $[P_1]...[P_{n_1}]$ <USER>	related/unrelated
	Emotion	$[M_1]...[M_{n_3}]$ [MASK] $[Q_1]...[Q_{n_2}]$ <CANDIDATE> $[P_1]...[P_{n_1}]$ <USER>	interesting/boring
	Action	$[P_1]...[P_{n_1}]$ <USER> [SEP] $[Q_1]...[Q_{n_2}]$ <CANDIDATE> [SEP] $[M_1]...[M_{n_3}]$ [MASK]	yes/no
	Utility	$[Q_1]...[Q_{n_2}]$ <CANDIDATE> $[M_1]...[M_{n_3}]$ [MASK] $[P_1]...[P_{n_1}]$ <USER>	good/bad
Hybrid Template	Relevance	$[P_1]...[P_{n_1}]$ <USER> [SEP] $[Q_1]...[Q_{n_2}]$ <CANDIDATE> [SEP] This news is [MASK] to the user's area of interest	related/unrelated
	Emotion	$[P_1]...[P_{n_1}]$ <USER> [SEP] $[Q_1]...[Q_{n_2}]$ <CANDIDATE> [SEP] The user feels [MASK] about the news	interesting/boring
	Action	$[P_1]...[P_{n_1}]$ <USER> [SEP] $[Q_1]...[Q_{n_2}]$ <CANDIDATE> [SEP] Does the user click the news? [MASK]	yes/no
	Utility	$[P_1]...[P_{n_1}]$ <USER> [SEP] $[Q_1]...[Q_{n_2}]$ <CANDIDATE> [SEP] Recommending the news to the user is a [MASK] choice	good/bad

Prompt4NR Templates	AUC	MRR	NDCG@5	NDCG@10
Discrete Template				
Relevance	68.77	33.42	37.20	43.36
Emotion	68.77	33.29	37.12	43.19
Action	68.76	33.22	37.02	43.26
Utility	**68.94**	**33.62**	**37.47**	**43.61**
Ensembling	69.34	33.76	37.71	43.80
Continuous Template				
Relevance	69.25	33.72	37.75	43.79
Emotion	68.76	33.51	37.39	43.47
Action	68.58	33.37	37.17	43.30
Utility	**69.10**	**33.96**	**37.91**	**43.92**
Ensembling	69.43	34.06	38.11	44.14
Hybrid Template				
Relevance	68.47	33.26	37.20	43.24
Emotion	68.59	33.26	37.19	43.29
Action	**69.37**	**34.02**	**37.96**	**44.00**
Utility	68.79	33.45	37.35	43.49
Ensembling	69.22	33.78	37.77	43.87

Module 5: Paradigm 2

`GLLM4Rec` (LLMs as Generators)

`GLLM4Rec`: The Core Idea

In this paradigm, we use the LLM's **generative capabilities** to *create* the recommendation itself.

The LLM *is* the recommender. It takes a unified prompt with task instructions, user history, and candidates, and **generates the final answer as text**.

`GLLM4Rec`: Taxonomy of Methods

These models are split into two families, based on whether the LLM's parameters are updated.

1. **Non-Tuning:** Use the pre-trained LLM as-is.
    - **Prompting (Zero-Shot):** Give the LLM a task instruction.
    - **In-Context Learning (Few-Shot):** Give the instruction + a few examples.

`GLLM4Rec`: Taxonomy of Methods

These models are split into two families, based on whether the LLM's parameters are updated.

2. **Tuning:** Update the LLM's parameters for the RecSys task.

`GLLM4Rec` (Non-Tuning)

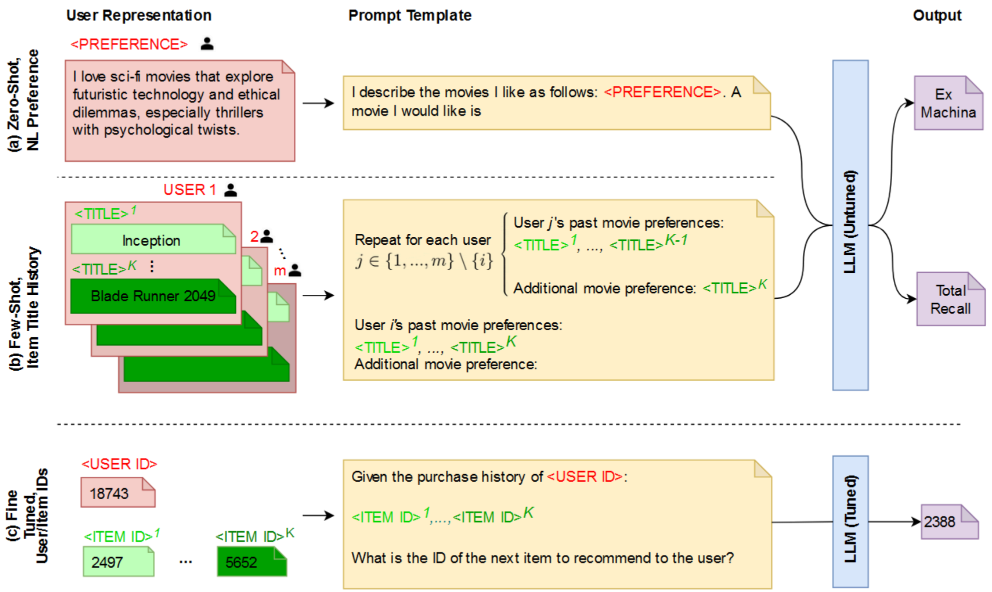
We can prompt LLMs to perform many RecSys tasks "out-of-the-box".

- **Task Examples:** We can ask for Top-K items, Rating Prediction, Conversational replies, or Explanation Generation.

`GLLM4Rec` (Non-Tuning)

We can prompt LLMs to perform many RecSys tasks "out-of-the-box".

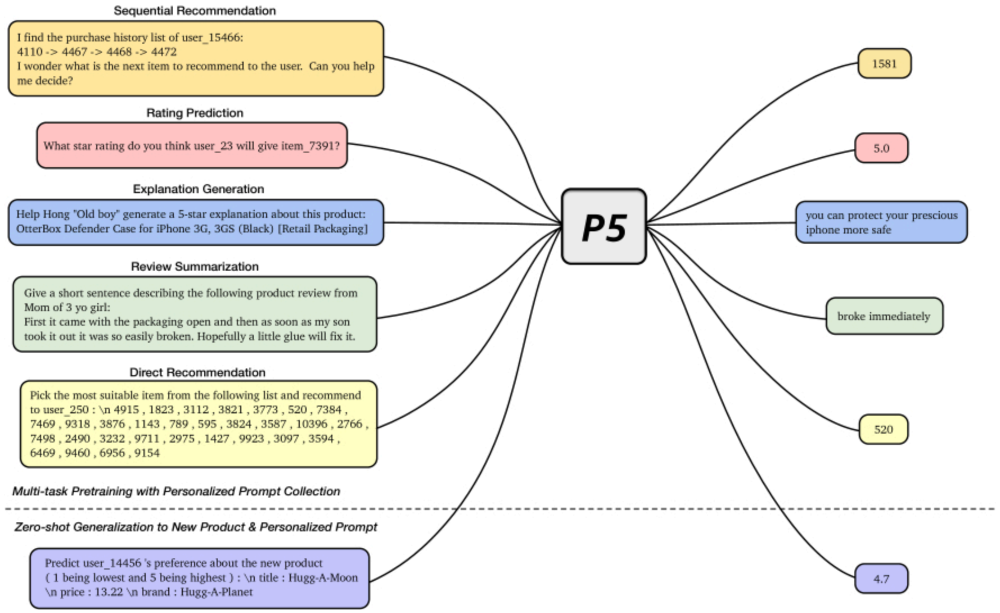
- **Prompt Examples:** We can prompt with...
  - **Natural Language Profiles**
  - **Item Title History**



`GLLM4Rec` P5[6]

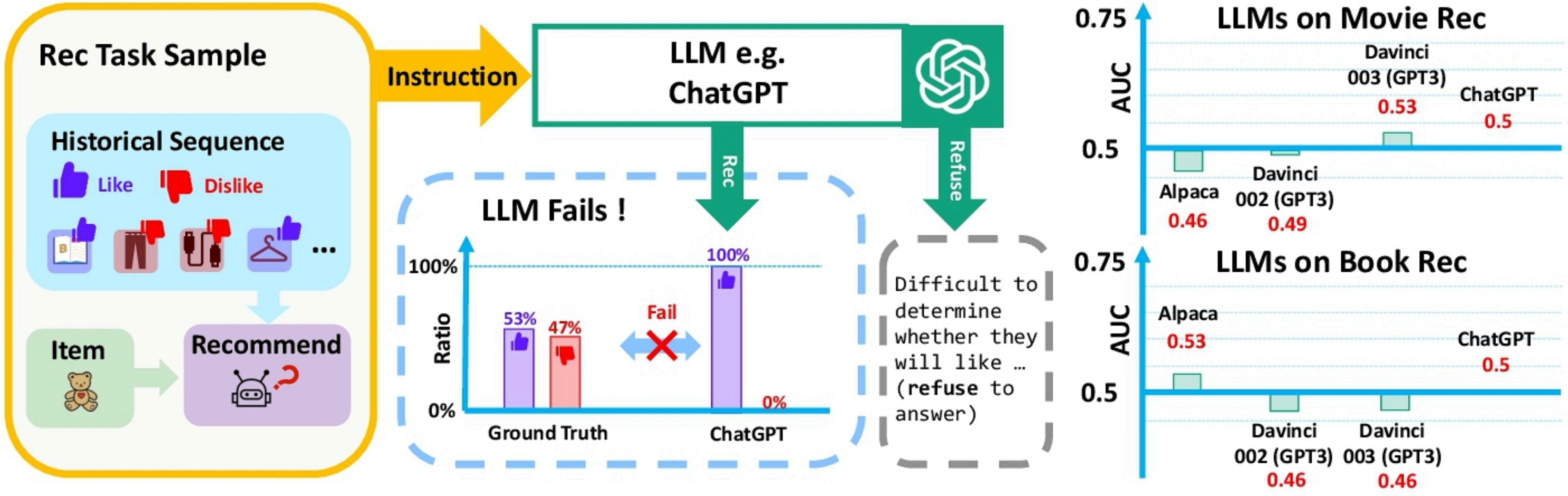
To improve performance, we can **Tune** the LLM on recommendation-specific data.

- **P5 (Instruction Tuning):**
  - *Unifies* all RecSys tasks into a single text-to-text framework.



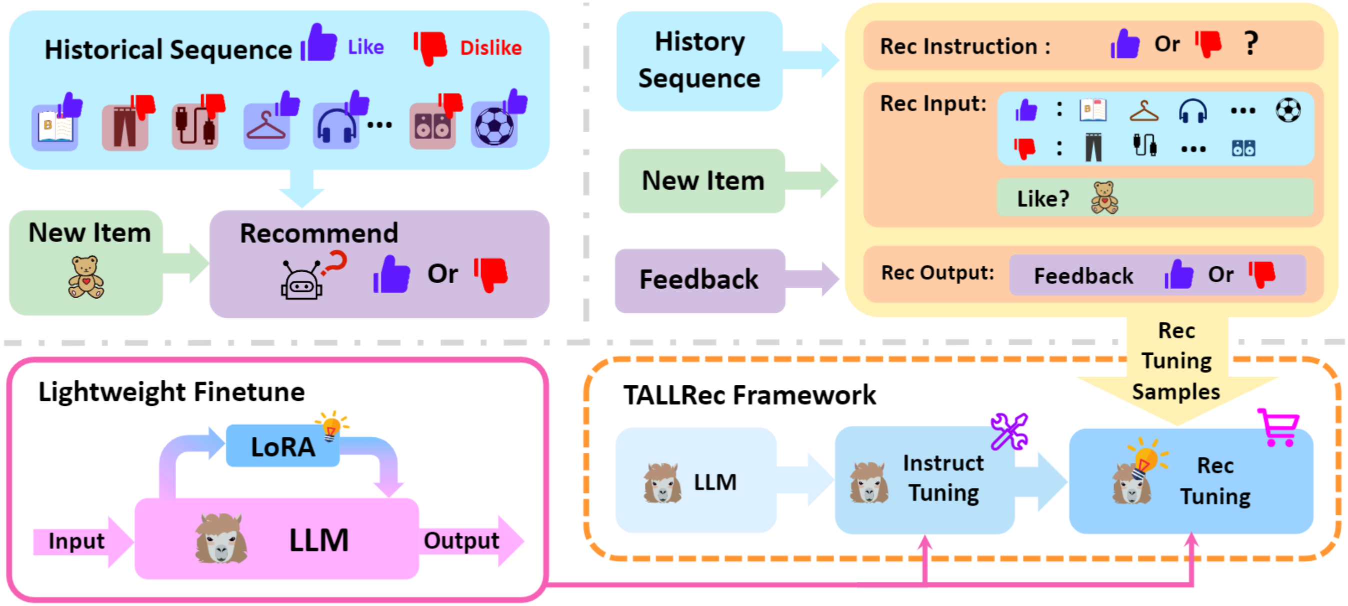
`GLLM4Rec` TALLRec[7]

- **TALLRec (Instruction Tuning):**



GLLM4Rec TALLRec

- **TALLRec (Instruction Tuning):**



Few-shot	GRU4Rec	Caser	SASRec	DROS	TALLRec
movie 16	49.07	49.68	50.43	50.76	**67.24‡**
movie 64	49.87	51.06	50.48	51.54	**67.48‡**
movie 256	52.89	54.20	52.25	54.07	**71.98‡**
book 16	48.95	49.84	49.48	49.28	**56.36**
book 64	49.64	49.72	50.06	49.13	**60.39‡**
book 256	49.86	49.57	50.20	49.13	**64.38‡**

Module 7: PhD Spotlight

[My Research] Empowering the "Genetator" with Knowledge

My Research: Empowering the "Genetator"

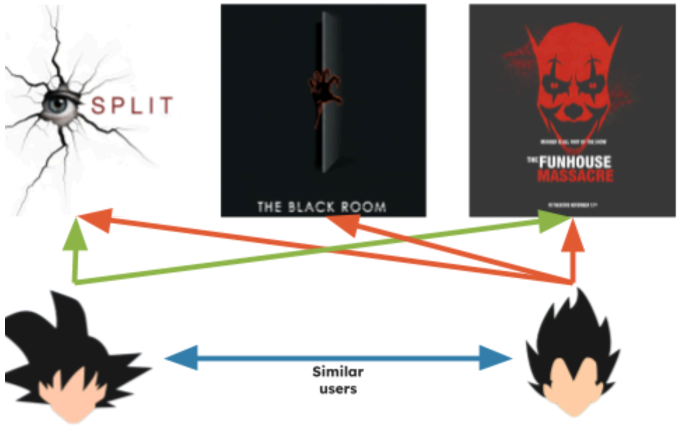
My Research (Petruzzelli et al., UMAP 2025) [9]:

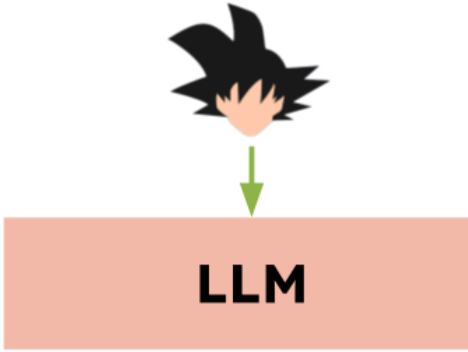
We provide a solution by **"injecting" domain-specific knowledge** directly into the LLM via fine-tuning.

My Research: Motivation & Hypothesis

Motivation: LLMs generate recommendations based on their *pre-trained knowledge*. But this knowledge may be:

1. **Incomplete:** The model may lack knowledge about *niche* items (e.g., indie music, specialized books).

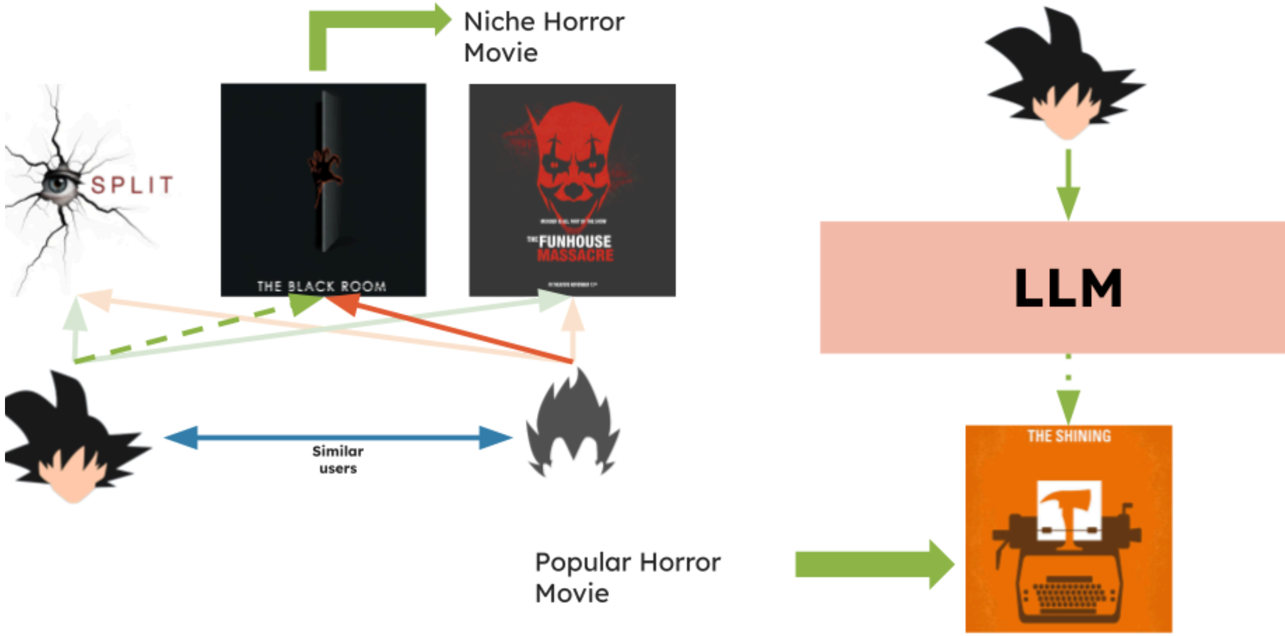




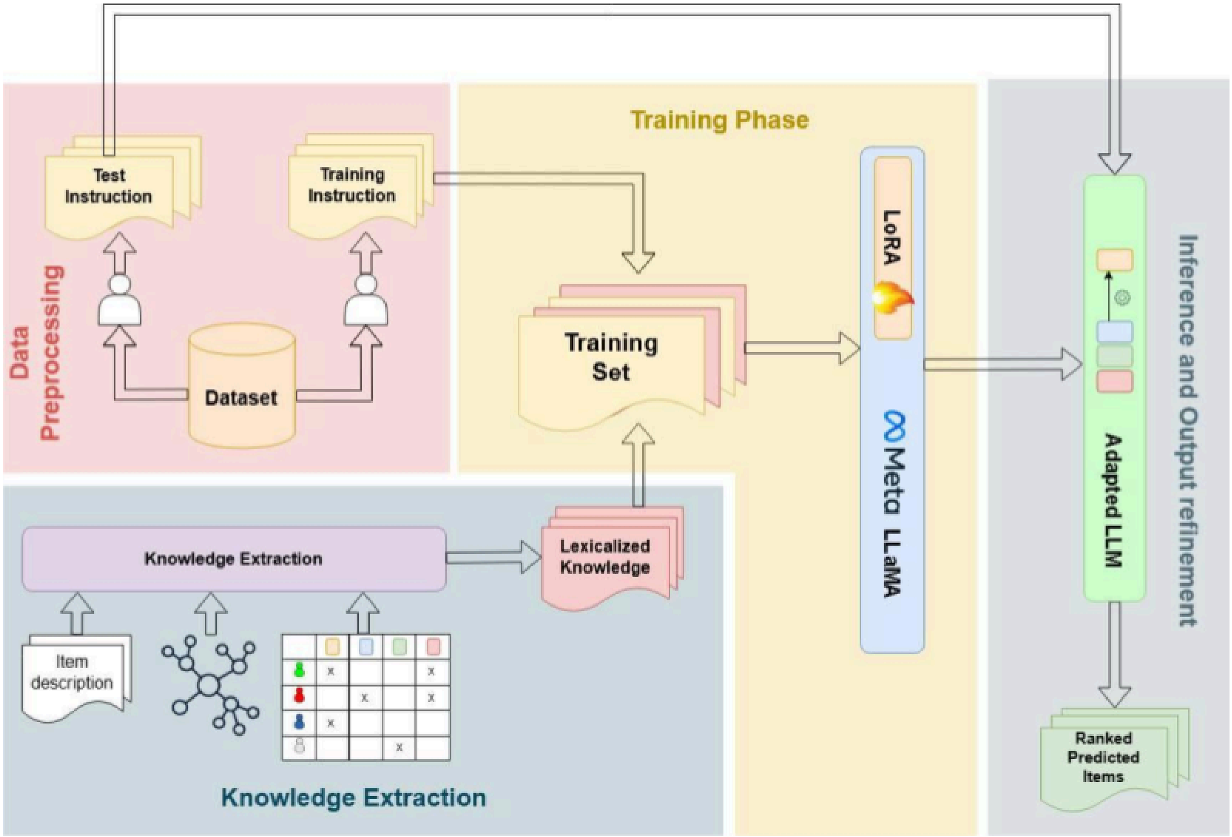
My Research: Motivation & Hypothesis

Motivation: LLMs generate recommendations based on their *pre-trained knowledge*. But this knowledge may be:

1. **Incomplete:** The model may lack knowledge about *niche* items (e.g., indie music, specialized books).



My Research: Methodology



My Pipeline: 1. Knowledge Extraction

We gathered domain-specific knowledge from three sources:

1. **Textual Data:**
   - The standard item descriptions, plots, etc.
2. **Knowledge Graphs (KGs):**
   - e.g., `(Home Alone 2, starring, Joe Pesci)`.
3. **Collaborative Data:**
   - e.g., `People who like {Item A} also like {Item B}`.

My Pipeline: 2. Lexicalization & Training

We **lexicalize** (turn into text) this structured knowledge so the LLM can read it.

Source	Lexicalized Knowledge
Text	`<begin_...>` Kevin McCallister is back... `<end_...>`
KG	`<begin_...>` Home Alone 2... Actors playing Joe Pesci... `<end_...>`
Collab.	`<begin_...>` People who like Home Alone 2... also tend to like The How the Grinch... `<end_...>`

We fine-tune the LLM (LLaMA 3 8B) on a **combined objective**

Key Finding 1 (RQ1)

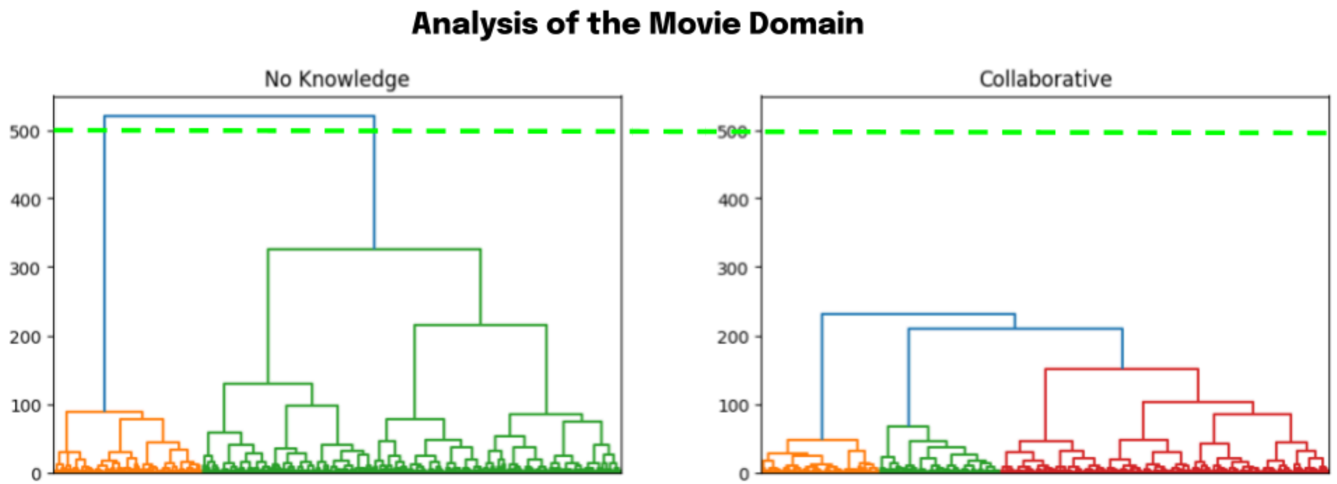
We compared injecting knowledge vs. no injection (baseline LLM).

- **Result (Music & Books):** Injecting knowledge (Text, KG, Collab) *improved* recommendation accuracy.
- **Result (Movies):** Injecting knowledge **did *not* improve accuracy**.

Domain	KN Source	P@5 ↑	R@5 ↑	NDCG@5 ↑	AvgPop@5 ↓
Movies	No Knowledge	**0.7654**	**0.2105**	**0.7728**	**0.0921**
	Text	0.7384	0.2063	0.7572	0.0915
	Graph	0.7534	0.2057	0.7607	0.0921
	Collaborative	0.7611	0.2070	0.7709	0.0927
Music	No Knowledge	0.8089	0.42723	0.8042	0.0390
	Text	**0.8428***	**0.4435***	**0.8491***	0.0393
	Graph	0.8259	0.4368	0.8276	0.0390
	Collaborative	0.8210	0.4341	0.8282	0.0384
Books	No Knowledge	0.7816	0.6317	0.8601	0.0113
	Text	0.7998	0.6494	0.8895	0.0111
	Graph	0.8018	0.6498	0.8869	0.0112
	Collaborative	**0.8049***	**0.6505***	**0.8920***	0.0114

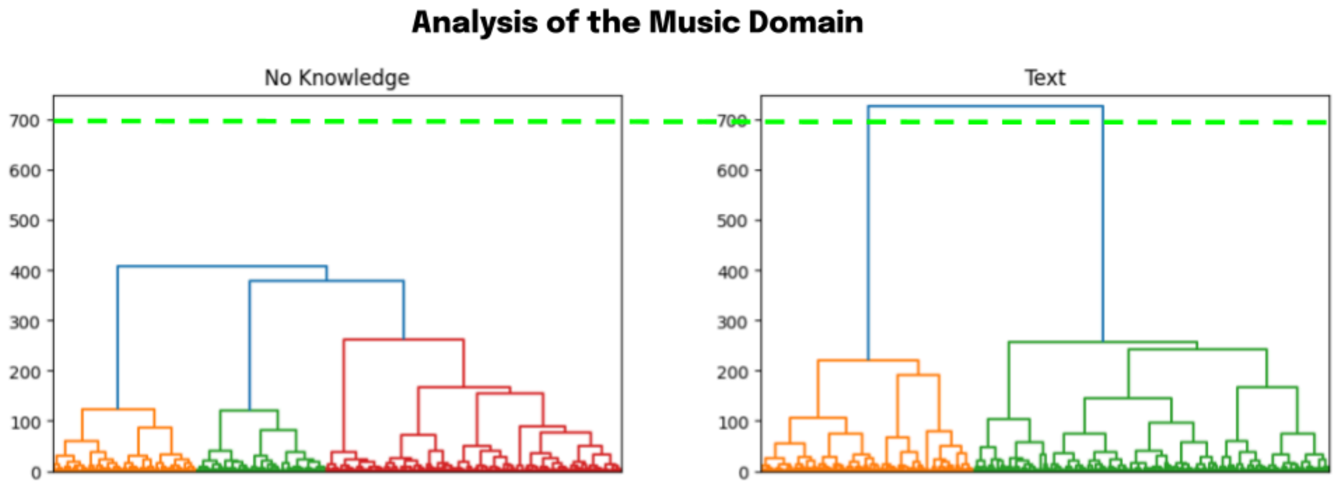
...Why? (The Conclusion)

This result proves our hypothesis.



...Why? (The Conclusion)

This result proves our hypothesis.



Module 6: Paradigm 3

LLMs as Conductors (The "Brain")

The "Conductor" Paradigm

The most advanced paradigm is not `LLM vs. RS`, but `LLM + RS`.

Here, the LLM acts as the "brain" or **"Conductor"** that *manages a process* and *coordinates* other, more specialized tools (like a retriever, a classic RS).

"Conductor" Example 1: RAG

Retrieval-Augmented Generation (RAG) is one way to fight hallucinations.

The LLM "conducts" the process:

1. **User Action:** User provides a query: "90's sci-fi thriller".
2. **Tool Call (Retriever):** The LLM sends the query to a **Retriever**, which searches a "Knowledge Corpus".
3. **Tool Output (Candidates):** The Retriever returns a *factual* list: ["The Matrix", "Inception", "The Bourne Identity"].
4. **LLM (Conductor):** The LLM's *only* job is to **re-rank this factual list**. It can't hallucinate items that don't exist.

"Conductor" Example 2: Conversational Agents

This is the most complex "conductor" role.

The LLM "conducts" a full, multi-turn dialogue by:

1. **Classifying Intent:** "What does the user want.
2. **Updating State:** "What have I learned?" (e.g., `cuisine_type: "Japanese"`) .
3. **Selecting Actions:** "Should I call a tool? Or ask a question.
4. **Generating a Response:** Synthesizing all info into a reply.

"Conductor" Example 3: RecLLM [10]

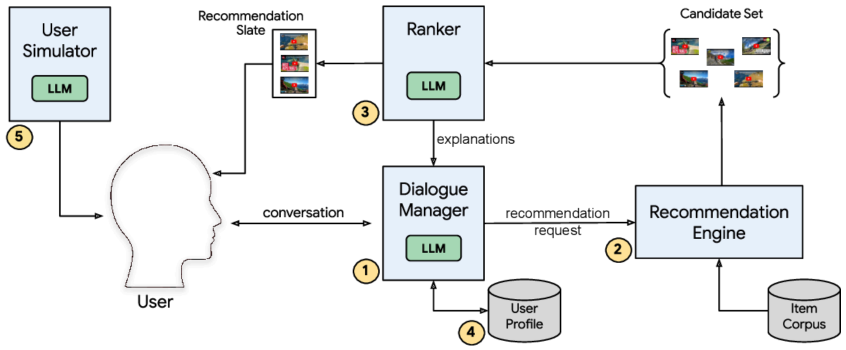
RecLLM proposes a roadmap for an end-to-end Conversational RecSys.

- **Goal:** Use LLMs for *every* part of the stack: understanding, dialogue, and explanation.
- **Key Challenge:** Lack of conversational training data.
- **Solution:** Use an LLM to build a **User Simulator** to generate synthetic conversations.

RecLLM: Architecture

The system is modular, with the LLM acting as the core reasoning engine.

1. **Dialogue Manager:** Maintains conversation state.
2. **User Profile:** Interpretable natural language profile.
3. **Retriever:** Fetches candidates from a large corpus (YouTube videos).
4. **Ranker & Explainer:** Selects items and generates explanations.



RecLLM: The User Simulator

How do you train a CRS without data? **Simulate it.**

- **User Simulator:** An LLM prompted with a specific "persona" and "goal".
- **System:** The RecLLM model.
- **Loop:** They talk to each other.
- **Result:** Thousands of synthetic dialogues used to fine-tune the production model.

Part 3: Evaluation, Risks, & Grand Challenges

"Do they work?" and "Are they safe?"

Module 8: Evaluating the "Un-evaluable"

Metrics for Generative RecSys

The Evaluation Challenge

Evaluating `SASRec` is easy:

- **Prediction:** `item_4`
- **Ground Truth:** `item_4`
- **Result:** Correct (Hit@1 = 1)

But how do you evaluate a **generative model**?

- **Prediction:** "Based on your love of classics, I suggest 'The Great Gatsby' movie."
- **Ground Truth:** `item_1234`
- **Result:** ???

A New Toolkit of Metrics (Offline)

When the output is a **list** (e.g., in my research), we still use the classics:

- **Precision@k, Recall@k, nDCG@k**

When the output is **text** (e.g., an explanation):

- **NLP Metrics:** We borrow from NLP.
  - **BLEU:** Measures *precision* of n-gram overlap .
  - **ROUGE:** Measures *recall* of n-gram overlap .
  - **Perplexity:** Measures *fluency* and model confidence (lower is better).

"LLM-as-a-Judge"

Use GPT-4 as your evaluator.

- **Method:** Feed the generated output to GPT-4 with a detailed rubric .
- **Prompt:** "Score this explanation from 1-10 on 'helpfulness' and 'factuality'."
- **Findings:** This is surprisingly highly correlated with human evaluators (e.g., 80% agreement).
- **Risks:** This method has known biases:
  - **Position Bias:** Prefers the *first* option it's shown.
  - **Verbosity Bias:** Prefers *longer*, more verbose answers.
  - **Self-Enhancement Bias:** Prefers answers generated by itself.

Online Evaluation

Offline metrics are just a proxy. The only ground truth is real user behavior.

- **A/B Testing:** This is the gold standard for online evaluation .
  - **Control (A):** The old recommender.
  - **Treatment (B):** The new Gen-RecSys model.
  - **Metrics:** We measure real business outcomes: Click-Through-Rate (CTR), Add-to-Cart, Session Length, Task Completion Time .

SOTA Eval: Simulation-based Evaluation

Online A/B tests are the ground truth, but they are **slow and expensive**.
A new alternative: **Simulate user behavior with LLM-based agents** .

Why LLMs as Users?

- They understand natural language, can adapt to scenarios, and can "reason" about choices, making them realistic proxies for human users.

Example 1: Simulating Search (USimAgent) [8]

- An LLM agent is used to simulate user search patterns, such as "querying, clicking, and stopping behaviors".

Part 4: Building a Transformer Recommender

https://tinyurl.com/Bert4Rec

Questions?

Thank You!

- **Alessandro Petruzzelli**
- **Email:** alessandro.petruzzelli@uniba.it

References

1. **GRU4Rec:** Hidasi, B., Karatzoglou, A., Baltrunas, L., & Tikk, D. (2015). Session-based Recommendations with Recurrent Neural Networks. ICLR 2016.
2. **SASRec:** Kang, W. C., & McAuley, J. (2018). Self-Attentive Sequential Recommendation. ICDM 2018.
3. **BERT4Rec:** Sun, F., et al. (2019). BERT4Rec: Sequential Recommendation with Bidirectional Encoder Representations from Transformer. CIKM 2019.
4. **U-BERT:** Qiu, Z., et al. (2021). U-BERT: Pre-training User Representations for Improved Recommendation. AAAI 2021.

References

5. **Prompt4NR:** Zhang, Z., & Wang, B. (2023). Prompt Learning for News Recommendation. SIGIR 2023.
6. **P5:** Geng, S., et al. (2022). Recommendation as Language Processing (RLP): A Unified Pretrain, Personalized Prompt & Predict Paradigm (P5). RecSys 2022.
7. **TALLRec:** Bao, K., et al. (2023). TALLRec: An Effective and Efficient Tuning Framework to Align Large Language Model with Recommendation. RecSys 2023.
8. **USimAgent:** Wang, L., et al. (2024). USimAgent: Large Language Models for Simulating Search Users. SIGIR 2024.

References

9. **Petruzzelli et al.:** Petruzzelli, et al. 2025. Empowering Recommender Systems based on Large Language Models through Knowledge Injection Techniques. UMAP '25.
10. **RecLLM:** Friedman, L., et al. (2023). Leveraging Large Language Models in Conversational Recommender Systems. arXiv preprint.