# 第 2 章 推荐系统架构

从第 1 章的介绍中我们已经了解到，工业级推荐系统需要在海量用户并发访问的场景下，对用户请求进行毫秒级甚至亚秒级响应。这不仅要求系统具备极高的计算效率和吞吐能力，同时也对系统的稳定性、可扩展性以及实时更新能力提出了严格要求。

在实际业务中，推荐系统往往并非服务于单一场景，而是需要同时支撑短视频、直播、电商、广告、社区内容等多种业务形态。不同业务在内容形式、用户行为特征以及优化目标上存在显著差异，因此工业界逐渐演化出多业务协同的混合推荐架构（Hybrid Recommendation Architecture），以统一承载和调度不同业务流量。在具体业务内部，为了兼顾推荐效果与计算成本，绝大多数系统仍采用级联式漏斗架构，通过召回、粗排、精排、重排等多个阶段逐层筛选候选内容。

与此同时，推荐系统的优化目标也在不断扩展。传统推荐系统主要围绕用户需求展开，即从海量内容中寻找最适合当前用户的候选内容。然而，在内容平台场景下，仅从用户侧视角进行优化往往难以满足内容生态建设、创作者扶持以及供给侧增长等业务诉求。因此，工业界进一步发展出了以内容创作者为中心的作者侧推荐系统（Producer-to-User，P2U），形成了与用户侧推荐系统相互补充的双边推荐体系。此外，为了应对流量高峰、资源受限以及异常故障等情况，推荐系统通常还会设计多层降级机制，通过牺牲部分推荐效果换取整体服务的稳定性与可用性。

本章将重点介绍工业界推荐系统的整体架构设计，并从混合推荐系统、级联式推荐架构、作者侧推荐系统以及降级推荐系统等多个角度，分析不同架构产生的背景、核心思想及其适用场景。首先，我们从工业界最常见的混合推荐系统架构开始展开讨论。

## 2.1 混合推荐系统架构

从图 1.3 可以看到，在一次完整的推荐请求链路中，推荐系统通常并不直接面向用户提供服务，而是处于整个系统架构的后端位置。用户首先通过客户端发起请求，请求经过服务端处理后才会进入推荐系统。服务端除了负责网络通信与请求转发之外，往往还承担用户状态管理、业务逻辑执行、权限校验以及内容安全审核等大量非推荐相关功能。因此，在现代互联网架构中，推荐系统通常作为一个独立服务存在，并通过标准化接口与服务端进行交互。

在工程实践中，服务端与推荐系统之间通常会增加一层统一的接入层，该模块被称为 **Router 服务（推荐路由服务）**。Router 服务的核心职责是屏蔽推荐系统内部复杂的业务逻辑，对上提供统一接口，对下负责请求分发、流量调度与降级、结果聚合以及故障兜底隔离等工作。通过 Router 层的存在，服务端无需感知推荐系统内部的具体实现细节，从而实现推荐系统与业务系统之间的解耦。

然而，对于大型内容平台而言，用户一次请求所面对的往往并非单一推荐系统。以短视频平台为例，同一个首页信息流中可能同时出现短视频、直播间、电商商品、广告、图文内容以及平台运营活动等多种不同类型的内容。由于这些内容在数据来源、业务目标、模型体系以及优化指标上均存在显著差异，因此通常由不同的推荐系统独立负责。为了在统一的产品界面中向用户呈现这些异构内容，平台需要将多个业务推荐系统整合到同一套服务框架之下。这种由多个业务推荐系统协同工作的架构被称为**混合推荐系统（Blended Recommendation System）**。

需要特别说明的是，这里的“混合（Blended）”并非传统推荐算法中“多路召回融合”或“多模型融合”的含义，而是指多个业务线推荐系统在平台架构层面的统一整合。例如短视频推荐系统、直播推荐系统、电商推荐系统以及广告推荐系统各自独立运行，但最终共同参与同一次用户请求的推荐结果生成。

因此，混合推荐系统关注的问题已经不再是单一内容的排序问题，而是一个更高层次的平台资源分配问题：

**定义 2.1**

在有限的曝光资源和展示槽位约束下，如何在不同业务之间合理分配流量，并实现平台整体收益最大化。♣

从本质上看，混合推荐系统已经从传统的“内容排序问题（Ranking Problem）”演化为“资源分配问题（Resource Allocation Problem）”。在实际工业系统中，混合推荐架构往往十分复杂。以短视频平台为例，平台同时存在短视频、直播、电商、广告、图文等多个业务域，每个业务域都拥有独立的召回、排序以及策略体系。当用户发起一次推荐请求时，不同业务系统会并行地产生各自的推荐结果，这些结果最终汇聚形成一个异构候选集合（Heterogeneous Candidate Set），随后再经过统一的混排与资源分配流程生成最终推荐列表。

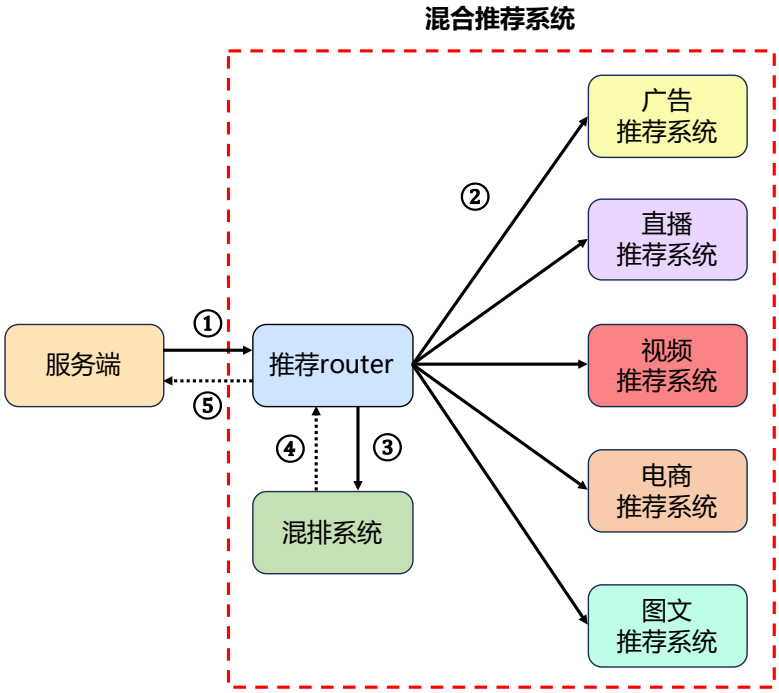


图 2.1: 服务端与混合推荐系统之间的系统架构与调用关系。

用户的一次推荐请求，从服务端进入推荐系统再返回结果的整体流程如图 2.1 所示，大致可以分为以下五个步骤：

1. 服务端向推荐 Router 发起 RPC 请求，请求获取推荐结果；
2. Router 接收到请求后，根据业务配置和流量策略，并行调用多个业务推荐系统；
3. 各业务推荐系统分别完成召回、排序等流程，并返回候选结果；
4. Router 收集各业务返回结果后，将候选集合发送至混排服务（Blending Service）；
5. 混排服务完成跨业务资源分配与排序优化，生成最终推荐列表，并经 Router 返回给服务端。

需要说明的是，图 2.1 实际上是图 1.3 中推荐链路的进一步展开，其目的是突出混合推荐系统中的关键模块与调用流程。在真实工业系统中，架构复杂度远高于图中展示的内容。除了业务推荐系统和混排服务之外，通常还会引入模型推理服务、实时特征服务、高性能缓存、用户行为序列存储、分布式索引等模块，共同参与推荐决策过程。为了便于理解，本节仅保留最核心的调用链路。

在混合推荐架构下，平台往往同时承担多种业务目标。例如：

- 短视频业务关注用户观看时长和用户留存；
- 直播业务关注直播间点击率、观看时长以及打赏收入；
- 电商业务关注点击率、成交转化率和 GMV；
- 广告业务关注广告收益和广告主 ROI。

上述的这些业务目标既互相关联，又存在天然的冲突。例如，广告曝光能够提升平台营收，但过多广告会损害用户体验，导致 app 时长大幅下跌；电商内容有助于提升交易规模，但也可能降低用户内容消费时长。因此，平台层面需要在有限曝光资源下平衡不同业务的利益诉求。由此就会带来的核心挑战主要包括：

- **目标冲突（Objective Conflict）**：不同业务追求的优化目标并不一致，甚至存在相互竞争关系；
- **资源竞争（Resource Competition）**：推荐位和曝光机会有限，不同业务需要竞争同一资源池；

- **生态公平（Ecosystem Fairness）**：需要保证不同业务、不同创作者以及不同内容类型获得合理曝光机会；
- **长期收益与短期收益平衡（Long-term vs. Short-term Trade-off）**：避免为了短期收益而损害平台生态和用户长期价值。

因此，对于工业界推荐算法工程师而言，优化目标已经不仅仅是提升单个业务指标，而是需要站在平台整体视角思考问题。一个业务指标的局部最优，并不一定意味着平台整体收益的最大化。例如，广告系统单独提升收入可能导致用户留存下降，而短视频系统单独追求时长增长也可能挤压电商和直播等业务的发展空间。

从数学角度看，混合推荐系统本质上可以被建模为**带约束的多目标优化问题（Constrained Multi-objective Optimization Problem）**。系统需要在曝光资源、业务配额、公平性约束以及用户体验约束等条件下，同时优化多个业务目标，并寻求平台层面的全局最优解。

在实际工业系统中，这种全局优化通常由混排服务承担。混排服务不仅负责融合不同业务返回的候选结果，更重要的是通过流量配额、公平竞价、多目标优化以及约束求解等机制，在平台层面完成资源分配决策，从而引导整个推荐系统向全局最优方向演化，避免陷入单一业务驱动的局部最优，最终实现用户价值、商业价值与生态价值之间的动态平衡。

## 2.2 级联式推荐系统

从推荐系统整体优化的视角来看，工业界的大型内容平台通常采用混合推荐系统（Blended Recommendation System）架构，以统一管理短视频、直播、电商、广告等多个业务之间的流量分配与资源协调问题。然而，如果将视角进一步聚焦到某一个具体业务内部，例如短视频推荐、直播推荐或商品推荐，那么其核心任务则转变为：如何从海量候选内容中快速、准确地找到当前用户最感兴趣的内容。在这一场景下，工业界主流采用的是**多阶段级联式推荐系统架构（Multi-stage Cascade Recommendation System Architecture）**。

从第 1 章的分析可以看出，现代互联网平台通常需要同时面对亿级用户规模与十亿级内容规模所带来的双重挑战。对于一次推荐请求而言，系统既要保证推荐结果具有足够高的个性化程度，又必须在几百甚至几十毫秒内完成整个计算过程。如果直接对全量内容进行复杂模型打分，无论从计算资源还是响应延迟角度来看都是不可接受的。

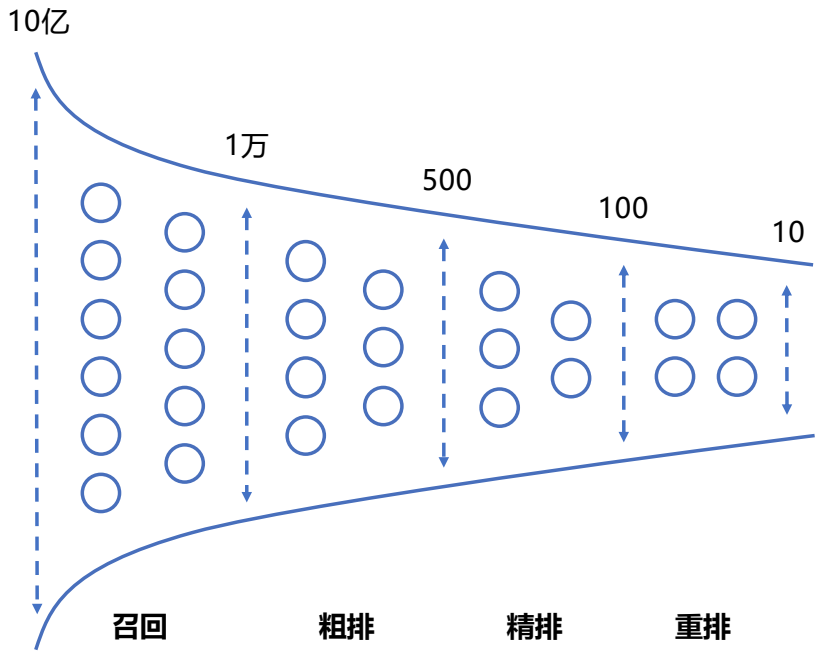


图 2.2: 推荐系统级联式架构示意图。

因此，工业界推荐系统逐渐演化出一种典型的“漏斗式（Funnel-based）”架构：在推荐流程的前期使用计算代

价较低的方法快速缩小候选集规模，在后期逐步引入更复杂、更精确的模型进行深度评估。这种设计使得系统能够在有限的计算资源约束下，实现推荐效果与服务性能之间的最佳平衡。从本质上看，级联式推荐系统是一种**逐层筛选（Progressive Filtering）**与**逐层精炼（Progressive Refinement）**的过程。随着推荐流程不断向后推进，候选集合规模持续减小，而模型复杂度和预测精度则持续提升，最终实现从海量内容到少量高质量推荐结果的逐级收敛。

如图 2.2 所示，现代推荐系统通常采用四阶段级联架构，整体流程可划分为：

召回（Retrieval）→ 粗排（Pre-Ranking）→ 精排（Ranking）→ 重排（Re-Ranking）

这一架构已经成为工业界推荐系统的事实标准，其核心思想是在不同阶段采用不同复杂度的算法，实现效率与效果的最优平衡。

### （1）召回阶段：从十亿内容中找到一万个候选

召回阶段（Retrieval Stage）是整个推荐链路的入口，其主要目标是在极短时间内从海量内容库中筛选出与用户兴趣相关的候选内容。对于大型内容平台而言，候选物品规模往往达到数十亿甚至上百亿级别。如果直接进入排序阶段，计算成本将呈指数级增长。因此，召回阶段必须首先完成一次大规模的候选压缩。

工业界通常采用**多路召回（Multi-channel Retrieval）**策略，不同召回通路分别从不同角度刻画用户兴趣，例如：协同过滤召回（Collaborative Filtering Retrieval），Embedding 向量召回（ANN Retrieval），内容召回（Content-based Retrieval），热门召回（Popular Retrieval），关注关系召回（Social Retrieval），序列兴趣召回（Sequential Retrieval）等。每一路召回独立生成候选集合，最终汇总形成约 1 万至数万规模的候选池。在召回之后，系统通常还会执行一个过滤（Filtering）模块，用于完成：内容去重、已曝光内容过滤、风险内容过滤、用户黑名单过滤、合规与安全策略过滤、上下文约束过滤等。经过这一阶段处理后，整个推荐系统会获得一个相对干净且具备基本相关性的候选集合。

### （2）粗排阶段：用低成本模型快速筛选

经过召回过滤阶段之后，候选集合规模虽然已经大幅下降，但通常仍然包含数千到数万个候选内容。此时如果直接使用复杂深度模型进行打分，依然会产生巨大的计算开销。因此系统会引入粗排阶段（Pre-Ranking Stage），利用计算代价较低的模型进行快速筛选。粗排模型通常具有以下特点：模型结构简单、特征数量较少、推理速度快、支持大规模并发请求等特点。工业界常见的粗排模型包括：双塔模型（Two-Tower Model）、浅层 DNN、多塔模型等。粗排阶段不仅承担候选筛选任务，还经常承担业务流量调控职责，例如长尾内容扶持、新内容冷启动、内容垂类配额控制、创作者流量保护等。最终，粗排阶段会从数千至上万个候选中保留约数百个高质量候选进入下一阶段。

### （3）精排阶段：决定推荐效果的核心环节

如果说召回决定了系统是否能够“找到正确的内容”，那么精排阶段（Ranking Stage）则决定了系统能否“把最合适的内容排在最前面”。精排是整个推荐系统中最核心、最重要的模块，也是算法创新最活跃的部分。由于候选规模已经被压缩到数百级别，因此系统可以使用更加复杂的模型结构，并引入更丰富的用户行为特征进行建模。

常见精排模型包括：Wide & Deep、DeepFM、xDeepFM、DIN、DIEN、DCN、Transformer-based Ranking Model、多任务学习模型（MMOE、PLE）等。在这一阶段，精排模型会综合考虑：用户长期兴趣、用户短期兴趣、实时上下文、内容质量、作者特征、用户与内容交互历史。与此同时，现代推荐系统往往不再只优化 CTR，而是同时优化 CTR、CVR、观看时长、互动率、留存率等多个目标。经过精排后，系统通常会保留约 100 个左右的候选内容进入重排阶段。

### （4）重排阶段：从单点最优到列表最优

前面的召回、粗排和精排，本质上都在解决一个问题：

“这个内容对用户有多大吸引力？”

然而，当多个内容共同出现在推荐列表中时，仅仅优化单个内容的得分并不足够。例如：连续出现 10 条篮球视频、全部来自同一个创作者、全部属于同一种商品品类等。即使每条内容的预测点击率都很高，整体用户体验仍然可能较差。因此，重排阶段（Re-Ranking Stage）关注的不再是单个物品，而是整个推荐列表的质量。

重排系统通常会考虑：多样性（Diversity），新颖性（Novelty），公平性（Fairness），品类均衡（Category Balance），创作者生态治理，用户体验约束。从优化视角来看，重排实际上是在求解一个**序列级优化问题（List-wise Optimization Problem）**。其实现方式包括：列表级学习排序（List-wise Ranking）、Transformer 序列重排、强化学习重排等。最终，重排阶段会输出约 Top 10 的推荐结果，并返回上游调用方。

综上所述，多阶段级联推荐架构本质上是一种“先召回、再筛选、后优化”的分层决策体系。随着推荐流程不断推进，候选规模逐步缩小，而模型能力持续增强。通过这种由粗到精、层层过滤的设计思想，工业界成功解决了海量内容场景下推荐效果与系统性能之间的矛盾，使其成为现代推荐系统最核心、最经典的架构范式。

## 2.3 推荐系统的离线架构与在线架构

有了前面几节关于推荐系统系统位置、混合推荐架构以及业务目标的介绍，本小节我们进一步从工程实现的角度来理解推荐系统在工业界究竟是如何运作的。对于很多初学者而言，推荐系统往往等同于 CTR 预估模型、召回模型或者排序模型。然而在工业界，一个推荐模型只是推荐系统中的一个组成部分。一个真正可运行的推荐系统，还需要解决数据采集、样本构建、特征存储、模型训练、模型部署、在线推理以及服务稳定性保障等一系列工程问题。因此，本节将从**离线架构（Offline Architecture）**和**在线架构（Online Architecture）**两个部分展开介绍。这两个部分通常属于推荐系统架构工程师更加关注的内容，但对于未来从事推荐算法、推荐策略以及大模型推荐方向工作的读者而言，本节能够帮助大家从更加系统化的视角理解一个问题：

笔记 为什么推荐系统叫做“推荐系统”，而不仅仅叫做“推荐模型”？

答案在于：推荐模型只是系统中的一个组件，而推荐系统本质上是一套围绕用户行为数据不断运转、自我迭代的数据飞轮（Data Flywheel）。

### 2.3.1 离线架构

推荐系统的离线部分主要负责数据生产、样本构建、特征计算、索引维护以及模型训练等工作，其核心目标是为在线推荐系统持续提供高质量的数据和模型支撑。从数据流的角度来看，离线架构实际上承担着“学习用户兴趣”的职责：用户在产品中的每一次行为都会被记录下来，并最终转化为推荐模型训练所需要的监督信号。

如图 2.3 所示，推荐系统的离线架构主要包括推荐数据的获取与发送、用户反馈上传、推荐样本拼接、特征生成以及模型离线训练等环节。整个流程包含两个关键分支：

- 当用户与推荐物品发生交互行为（如点击、点赞、分享等）后，客户端会立即将行为日志上报至埋点网关。埋点网关对数据进行鉴权、校验、限流及格式标准化处理后，将其以 Kafka 消息的形式发布。下游的 **“反馈标签消费者”**（一个基于 Java 实现的 Kafka 消费服务）会实时消费这些消息，并将用户反馈以 Key-Value 的形式写入 Redis 缓存。其中，Key 通常由 `请求 ID + 物品 ID` 构成，便于后续样本拼接服务高效查询。
- 在用户每次发起推荐请求时，服务端通过推荐 router 调用下游推荐系统。在召回与排序过程中，系统会从索引服务中获取物品的完整特征，并将本次请求的用户特征、物品特征及上下文信息打包为 Kafka 消息发送出去。下游的 **“样本拼接消费者”** 会消费该消息，并在一个固定（例如 20 分钟）或动态设定的拼接窗口内，尝试从 Redis 中查询对应的用户反馈。若在窗口期内成功匹配到反馈标签，则将特征与标签拼接为完整的训练样本 $(user, item, context, label)$，并写入新的 Kafka Topic，供离线模型训练任务消费。

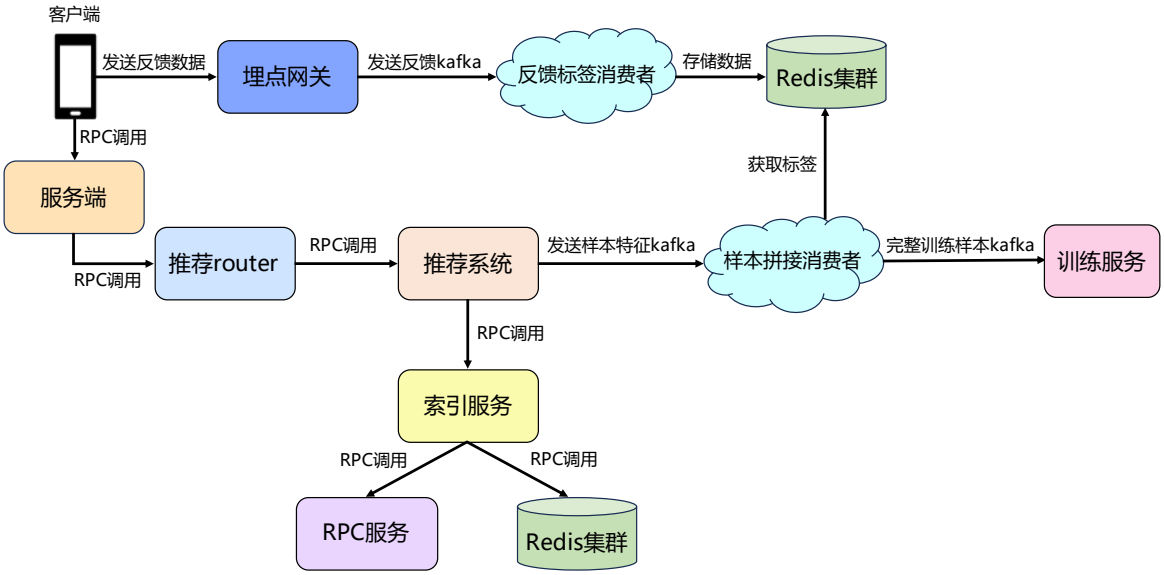


图 2.3: 推荐系统离线架构。

从本质上看，上述两个分支分别对应推荐系统中的**特征流（Feature Stream）**与**标签流（Label Stream）**。样本拼接服务则负责在合适的时间窗口内将二者关联起来，最终构建出模型训练所需的监督学习样本。以短视频场景为例，如图 2.4 所示，用户在观看视频时可以进行点赞、评论、分享等交互操作。当用户完成点赞或分享后，客户端会将这些行为以 JSON 格式上报至埋点网关。埋点网关处理后写入 Kafka，由反馈标签消费者写入 Redis；而样本拼接消费者则通过等待机制尝试拼接正样本，其核心目标是**最大化样本拼接率**，从而提升模型训练的数据质量与效果。

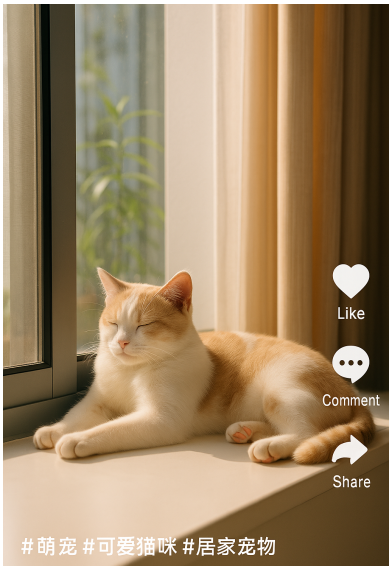


图 2.4: 短视频页面示意图。

在实际工业系统中，样本拼接率往往是离线架构的重要监控指标之一。拼接率过低意味着大量用户反馈无法与推荐请求对应起来，进而导致训练数据损失，最终影响模型效果。在图 2.3 所示的推荐系统离线架构中，涉及若干关键技术组件，它们共同支撑了用户行为数据的采集、传输、存储以及训练样本的构建过程。为了便于理解，本节对这些核心组件进行统一说明：

- **Kafka**：

  Kafka 是一个高吞吐、分布式的流式消息队列系统，广泛应用于日志采集、数据传输以及系统解耦。在推荐系统中，Kafka 主要负责承载用户行为日志（如曝光、点击、点赞、停留时长等）的实时传输，并将这

些数据准实时地发送至样本拼接、特征工程以及模型训练等下游模块。

- **索引服务**：
  此处的索引服务特指物品侧（Item-side）的特征索引服务，即根据 Item ID 快速获取对应物品的完整特征信息，例如 Embedding 向量、类别标签、统计特征以及内容属性等。该服务通常被在线推荐链路调用，因此也被称为正排索引（Forward Index）。需要注意的是，它与召回阶段使用的倒排索引或向量索引并不相同：后者用于大规模候选物品检索，而前者用于已知 Item ID 后的快速特征查询。
- **RPC 调用**：
  RPC（Remote Procedure Call，远程过程调用）是一种跨服务通信机制，允许调用方以本地函数调用的方式访问远程服务。在推荐系统中，用户画像服务、特征服务、索引服务以及模型推理服务之间通常通过 RPC 进行通信，从而满足在线推荐场景下低延迟、高并发的访问需求。
- **Redis 集群**：
  Redis 是一种基于内存的高性能键值存储系统，具备极低的读写延迟。通过分片（Sharding）和副本（Replication）机制，Redis 可以支持大规模水平扩展与高可用部署。在推荐系统中，Redis 常用于缓存用户特征、物品特征、实时统计指标以及样本拼接过程中的中间结果，从而提升整体系统性能。
- **样本拼接服务**：
  样本拼接服务负责将用户行为日志、用户特征、物品特征以及上下文信息等来自不同数据源的异构数据进行关联与融合，最终生成标准化训练样本：$(user, item, context, label)$。样本质量直接决定模型训练效果，因此样本拼接服务通常被认为是推荐系统离线训练流水线中最关键的组件之一。

读者可能会注意到，图 2.3 中离线训练所依赖的数据源是实时 Kafka 消息流，而非传统推荐学习中常见的静态用户—物品交互矩阵。这一设计源于工业级推荐系统对于数据规模与实时性的双重要求：在短视频、电商等场景下，每日活跃用户数与内容规模极其庞大，难以通过显式交互矩阵存储全部行为数据。同时，为了及时捕捉用户兴趣的动态变化，系统也需要尽可能缩短从用户行为产生到模型训练的数据链路。

因此，越来越多的工业级推荐系统开始采用基于实时数据流的训练范式。当然，并非所有业务场景都必须采用流式训练。例如在广告推荐场景中，由于用户每日接触的广告数量相对有限，很多系统仍然采用基于 HDFS 等离线存储的批处理方式，按照小时级甚至天级周期进行模型训练。实际工程中，推荐算法工程师需要根据业务规模、反馈延迟以及实时性要求，在实时流训练与离线批训练之间进行权衡与选择。

从上述流程可以发现，推荐系统离线架构的核心任务不仅仅是生成训练样本，还需要长期维护用户和物品两侧的数据资产。只有保证用户特征与物品特征能够持续更新并保持新鲜度，推荐模型才能不断学习用户兴趣并适应内容生态的变化。因此，在工业级推荐系统中，除了样本拼接流程之外，还需要构建完善的用户特征存储体系与物品特征存储体系，以支撑后续的模型训练、在线推理以及索引更新等核心环节。

### 用户侧特征存储

除了样本构建之外，推荐系统还需要长期维护用户画像（User Profile）以及用户行为序列。在工业界推荐系统中，用户行为通常不会仅保存最近几次交互，而是会被**长期持久化存储**。例如在短视频场景中，一个高活跃用户一年内可能产生数万甚至数十万次观看行为；在电商场景中，用户数年的浏览、点击、加购以及购买记录也具有重要价值。因此，用户行为日志通常会存储于 HDFS、Hive、ClickHouse 等分布式存储系统中，形成完整的用户历史行为库。常见存储内容包括：点击序列（Click Sequence）、浏览序列（View Sequence）、点赞序列（Like Sequence）、评论序列（Comment Sequence）、分享序列（Share Sequence）、购买序列（Purchase Sequence）。

由于原始行为序列长度往往非常长，在线推荐阶段无法直接读取全部历史记录，因此工业界通常会离线构建不同粒度的用户特征，例如：最近 50 条行为序列、最近 7 天行为序列、最近 30 天兴趣统计、长期兴趣画像等。这些特征最终会被写入用户特征中心（User Feature Store），供在线推荐服务实时读取。从某种意义上来说，用户特征中心承担着推荐系统“长期记忆”的角色，它将用户过去数月甚至数年的行为历史压缩成推荐模型可以直接使用的特征表示。

### 物品侧特征存储与索引更新

与用户侧特征类似，推荐系统同样需要维护海量物品的特征信息。以短视频平台为例，一个视频可能包含如下信息：视频 Embedding、作者 Embedding、视频标签、视频分类、发布时间、点赞数、评论数、分享数、完播率、实时热度等等。这些信息通常以：$(item\_id, feature)$ 的形式存储于**分布式索引系统**中。这里的索引服务本质上可以理解为一个**超大规模分布式 KV 缓存系统**。当推荐系统获得候选物品 ID 之后，便可以通过 RPC 快速查询对应的物品特征。需要注意的是，这里所说的索引属于**正排索引（Forward Index）**，主要解决：

笔记 已知 Item ID，如何快速获取其完整特征？

而召回阶段使用的**倒排索引**或向量索引则解决：

笔记 已知用户兴趣，如何快速找到相关 Item？

二者共同构成了推荐系统最重要的数据基础设施。由于物品状态会持续发生变化，因此索引也需要不断更新。工业界通常采用：**Batch 全量更新**和 **Streaming 实时更新**相结合的方式维护索引。其中，全量更新通常由离线任务轮询周期性执行，例如每小时、每 6 小时或每天重建一次索引；实时更新则由 Kafka 消息流进行触发，例如用户点赞、评论、购买等行为发生后，实时更新对应物品的统计特征。这种“Batch + Streaming”的双重更新模式既保证了索引的一致性，又保证了内容的新鲜度（Freshness）与特征的实时性。

综上所述，离线推荐系统承担着数据生产、特征构建、样本生成、索引维护以及模型训练等工作。从用户行为产生，到训练样本构建，再到模型训练和索引更新，整个过程形成了推荐系统的数据闭环，并持续为在线推荐系统提供高质量的数据与模型支撑。

## 2.3.2 在线架构

如果说离线推荐系统的核心任务是通过用户行为数据不断学习用户兴趣，那么在线推荐系统的核心任务则是在极短时间内完成一次推荐决策。它直接面向用户请求运行，属于客户端 → 服务端 → 推荐 Router → 推荐系统主服务这一调用链路中的核心组成部分，主要负责从海量候选物品中快速筛选出用户当前最可能感兴趣的 Top-K 结果。

与离线架构相比，在线推荐系统更加关注三个核心问题：

- **低延迟（Low Latency）**
- **高可用（High Availability）**
- **高吞吐（High Throughput）**

对于大型互联网平台而言，每一次用户下滑页面、刷新信息流或进入推荐页面，都会触发一次新的推荐请求。对于日活跃用户数达到千万甚至上亿的平台来说，推荐系统每秒可能需要处理数十万甚至上百万次请求。因此，在线推荐系统不仅需要保证推荐结果的准确性，还必须保证服务能够稳定、高效地响应用户请求。

在实际工程中，推荐系统通常存在严格的耗时约束。例如，一次推荐请求的整体响应时间往往需要控制在数百毫秒以内，而召回、粗排、精排以及重排等各个阶段又会进一步拆分各自的延迟预算（Latency Budget）。因此，在线推荐系统中的所有模块都必须在有限的时间窗口内完成计算，否则便可能导致整体推荐请求超时，进而影响用户体验。

除了低延迟之外，高可用性同样是在线推荐系统设计中的重要目标。由于推荐系统直接承载用户请求，一旦服务发生故障，用户可能会出现推荐结果为空、页面无法刷新、内容加载缓慢等现象。在短视频、资讯、电商等高度依赖推荐的产品中，这类问题往往会直接影响用户留存与核心业务指标。

因此，工业界通常会在推荐系统中设计一系列**兜底机制（Backup Mechanism）**。例如当召回服务异常、排序服务超时或者模型推理服务不可用时，系统可以自动切换至预先准备好的高热内容池、运营内容池或者缓存推荐结果，并按照一定策略返回给用户。虽然这种方式无法达到个性化推荐的效果，但能够保证推荐服务的基本可用性，从而避免用户出现“刷不出内容导致页面卡住或者空白”等情况。

除了服务稳定性之外，推荐系统还需要面对巨大的算力成本压力。由于用户访问行为具有明显的潮汐特征，因此推荐系统的请求量通常会呈现显著的波峰与波谷现象。如图 2.5 所示，推荐系统在一天内的 QPS（Query Per

Second）往往会随着用户活跃度变化而发生明显波动。其中，波峰通常对应用户集中使用产品的时间段，而波谷则对应用户活跃度较低的时段。

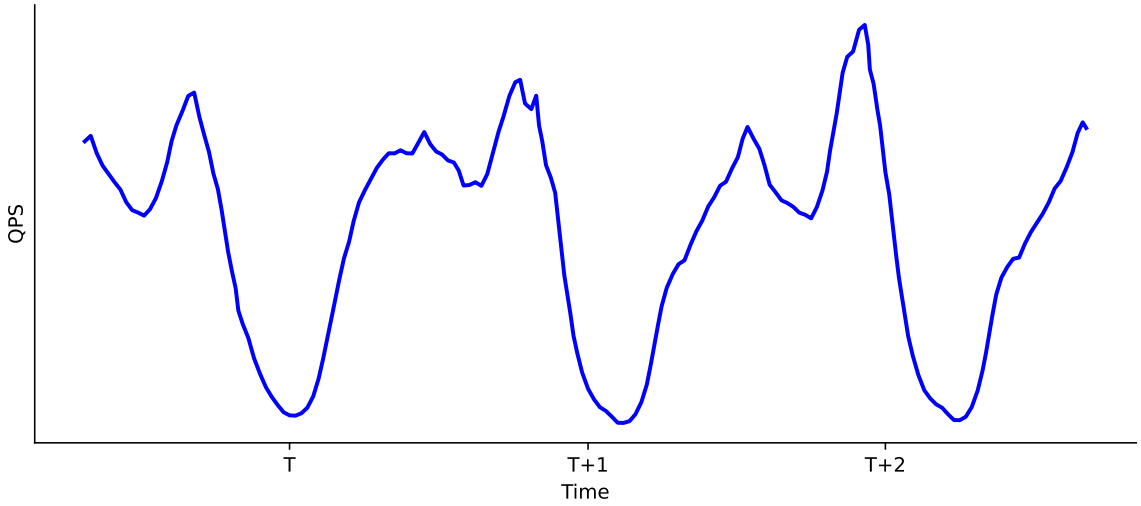


图 2.5: 推荐系统 QPS 高低峰期示意图。

对于推荐系统而言，峰值时段不仅意味着更高的流量压力，也意味着更大的算力消耗。如果始终采用最复杂的召回与排序策略处理所有请求，则需要投入大量服务器资源来应对短时间内的流量高峰，从而导致较高的基础设施成本。因此，在工业界的大规模推荐系统中，通常会根据实时流量情况进行**动态算力分配（Dynamic Resource Allocation）**。当系统处于低负载状态时，可以启用更复杂的召回策略、更大的候选集合以及更精细的排序模型；而在流量高峰阶段，则适当降低部分非核心计算开销，以保证整体系统的稳定运行。

在此基础上，许多互联网公司还会构建独立的**降级推荐系统（Degraded Recommendation System）**。所谓降级推荐，是指在资源紧张或系统压力过高时，采用一套计算成本更低但推荐效果略逊于实时推荐系统的方案。例如减少召回路数、缩小候选集合规模、关闭部分复杂模型，或者直接使用缓存好的推荐结果进行分发。虽然降级策略会带来一定程度的推荐效果损失，但能够显著降低服务器资源消耗，并保障核心业务指标的稳定性。

综上所述，在线推荐系统本质上是一个实时决策系统。它不仅需要在极短时间内完成候选召回与排序计算，还需要兼顾系统稳定性、资源利用率以及服务可用性等多方面要求。因此，工业界在线推荐系统的优化目标并不仅仅是提升模型效果，而是在推荐质量、响应延迟、系统稳定性以及算力成本之间寻求最优平衡。

## 2.4 降级推荐系统

读者在读到这一节的时候，难免会有一些疑问，即：

笔记 为什么需要降级推荐系统？什么是降级推荐系统？

在上一节中我们提到，推荐系统除了需要保证推荐效果之外，还需要兼顾系统稳定性与算力成本。当系统处于流量高峰期、计算资源紧张或部分核心服务发生异常时，如果仍然对所有请求执行完整的召回、粗排、精排与重排流程，往往会导致系统响应时间急剧上升，甚至出现服务不可用的情况。因此，工业界通常会设计专门的**降级推荐系统（Degraded Recommendation System）**，用于在资源受限场景下保障推荐服务的持续可用性。

在工业界推荐系统中，一种典型的降级策略是：

**定义 2.2**

在实时推荐系统正常运行时，将当前用户请求中排序结果中的 Top 10 至 Top 30 物品缓存至 Redis 中。当系统触发降级时，直接从 Redis 中读取这些预先计算好的推荐结果，并选取其中的前 10 个物品返回给用户，从而避免执行完整的召回与排序流程，大幅降低系统的计算开销和响应延迟。如果 Redis 缓存中还有没用完的缓存物品，下一次这个用户触发降级请求的时候还可以继续使用。

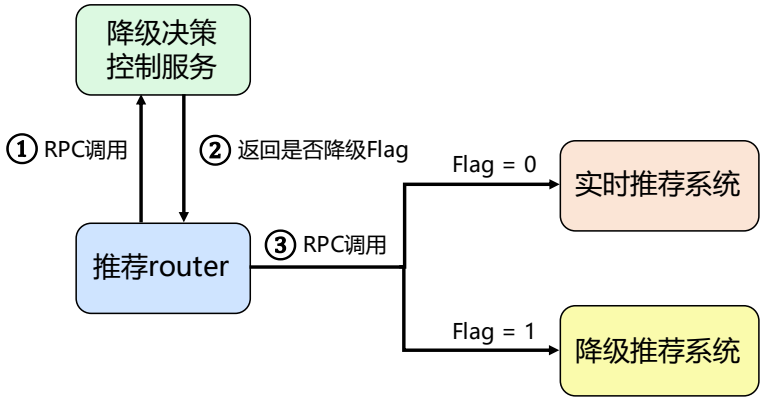


图 2.6: 降级与实时推荐系统架构图。

降级推荐系统与实时推荐系统的协同架构如图 2.6 所示，其整体决策流程如下：

1. 用户请求到达推荐 Router 后，Router 首先通过 RPC 调用**降级决策服务（Degradation Decision Service）**。
2. 降级决策服务根据当前系统状态返回是否触发降级的决策标识（Flag）。其中，Flag=0 表示继续使用实时推荐系统；Flag=1 表示进入降级模式。
3. 推荐 Router 根据返回结果选择后续处理链路：当 Flag=0 时，请求实时推荐系统；当 Flag=1 时，请求降级推荐系统，并返回降级推荐结果。

需要注意的是，图 2.6 中的降级决策服务本身也可以采用不同的实现方式。最简单的方法是基于规则进行控制，例如根据系统当前的 CPU 利用率、GPU 利用率、每秒请求数 (Query Per Second, QPS)、平均响应时间（Latency）等指标设置固定阈值。当系统负载超过预设阈值时自动进入降级状态；当系统恢复正常后再自动退出降级状态。例如，推荐 Router 及下游的服务在 CPU 利用率低于 75% 的安全线之内，我们能提前估计出整个推荐链路能够承载的平均 QPS 是 2 万。那么在每天的高峰期时段或者是节假日来临时，整个推荐系统的 QPS 往往会高于 2 万。这时推荐 Router 就可以随机选取 2 万之外的用户请求，让其路由到降级推荐系统，然后直接使用 Redis 降级缓存的结果进行返回。

除此之外，**降级决策也可以采用强化学习等智能决策方法实现**。此时，降级推荐系统不再基于固定规则，而是根据当前环境状态动态学习最优降级策略，以实现业务收益与资源消耗之间的最优平衡。关于强化学习在流量调控与资源分配中的应用，我们将在后续第 28 章中进行详细介绍。

在本节中，我们重点介绍一种经典的**动态算力分配框架**，即 DCAF（Dynamic Computation Allocation Framework）[2]。该方法发表于论文《DCAF: A Dynamic Computation Allocation Framework for Online Serving System》，其核心思想是：

**定义 2.3**

*打破传统推荐系统对所有请求“一视同仁”的算力分配方式，而是根据每个请求的潜在价值动态分配计算资源。*

在传统推荐系统中，多阶段级联架构（召回、粗排、精排等）通常采用固定计算预算。例如，精排阶段统一评估 200 个候选物品，无论用户价值高低均采用相同计算策略。这种方式虽然实现简单，但忽略了不同请求之间巨大的价值差异。例如，对于高活跃用户、高消费用户或者高商业价值用户而言，更复杂的模型计算往往能够带来更大的收益提升；而对于低价值流量，则未必值得消耗同样多的计算资源，可以直接使用降级推荐的结果。因此，从平台整体收益最大化的角度来看，不同请求理应获得不同等级的算力资源。DCAF 正是在这一背景下提出的动态算力分配框架。

形式上，DCAF 将资源分配问题建模为一个带约束的背包问题（Knapsack Problem）：在总计算资源预算 $C$ 的限制下，为每个请求 $i$ 从 $M$ 个候选动作中选择一个动作，以最大化整体收益：

$$
\begin{aligned}
\max_{x_{ij}} \quad & \sum_{i=1}^{N}\sum_{j=1}^{M} x_{ij} Q_{ij} \\
\text{s.t.} \quad & \sum_{i=1}^{N}\sum_{j=1}^{M} x_{ij} q_j \le C, \\
& \sum_{j=1}^{M} x_{ij} \le 1, \\
& x_{ij} \in \{0,1\},
\end{aligned}
\tag{2.1}
$$

其中：

- $x_{ij}=1$ 表示请求 $i$ 选择动作 $j$；
- $Q_{ij}$ 表示请求 $i$ 在动作 $j$ 下能够获得的预期收益；
- $q_j$ 表示动作 $j$ 所对应的计算资源消耗。

在广告推荐系统中，$Q_{ij}$ 通常采用预估 eCPM 进行衡量，而 $q_j$ 则对应不同精排策略的计算开销，例如需要评估的广告数量或模型推理成本。在短视频推荐系统中，$Q_{ij}$ 可以采用预估视频播放时长等指标来作为衡量依据，$q_j$ 表示实时推荐和降级推荐的算力消耗。

通过拉格朗日对偶分析，DCAF 进一步证明：在收益随计算资源增加而递增、但边际收益递减的条件下，最优策略可简化为：

$$
\arg\max_{j} \left(Q_{ij} - \lambda q_j\right) \tag{2.2}
$$

即对于每个请求，只需选择使

$$
Q_{ij} - \lambda q_j \tag{2.3}
$$

最大的动作即可。其中，$\lambda$ 为与全局资源预算相关的拉格朗日乘子，可通过二分搜索等方法高效求解。

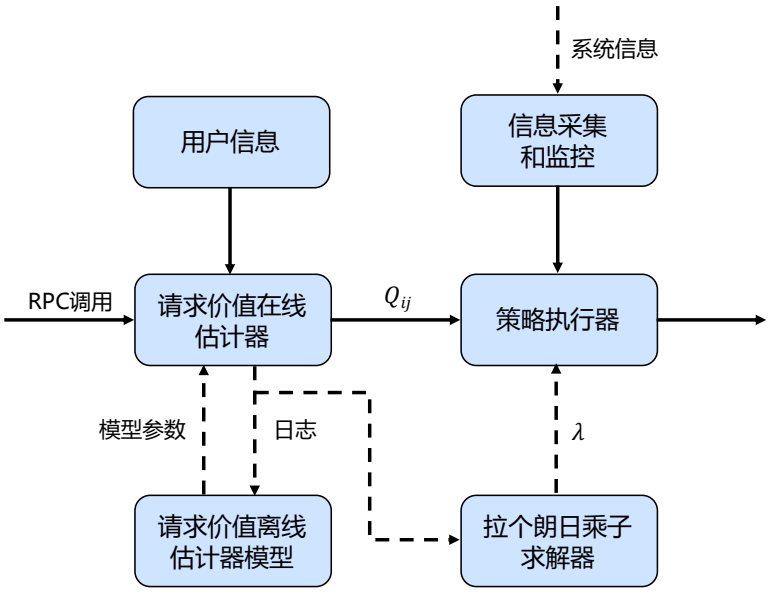


图 2.7: DCAF 系统架构示意图。

如图 2.7 所示，DCAF 在线服务系统主要由两个模块组成：

- **请求价值估计器（Offline Estimator）**

  利用历史日志训练轻量级价值评估模型，实时预测请求在不同动作下的预期收益 $Q_{ij}$。在广告场景中，该

收益通常采用 eCPM 衡量。为了降低线上开销，该模块通常复用已有推荐模型或广告模型中的中间特征。

- **策略执行器（Online Decision Maker）**

  根据价值估计结果以及当前最优 $\lambda$ 参数，为每个请求动态选择对应动作，实现实时算力分配。此外，DCAF 还引入了 `MaxPower` 机制。当系统遭遇流量突发（例如双十一大促）时，系统通过 PID 控制器实时监控 GPU 利用率、响应时间以及服务失败率等指标，并动态调整单请求允许使用的最大算力，从而保证整体系统的稳定运行。

论文实验表明，DCAF 已成功应用于淘宝展示广告系统。在广告收入（RPM）和点击率（CTR）基本保持不变的前提下，系统能够降低约 20% 的 GPU 资源消耗，同时显著提升高并发场景下的稳定性。从本质上来看，传统降级系统是一种“全局降级”策略，而 DCAF 则实现了“按请求价值进行差异化降级”的动态资源调度机制。当资源受限时，系统优先保障高价值流量的推荐质量，从而在业务收益与系统成本之间取得更优平衡。

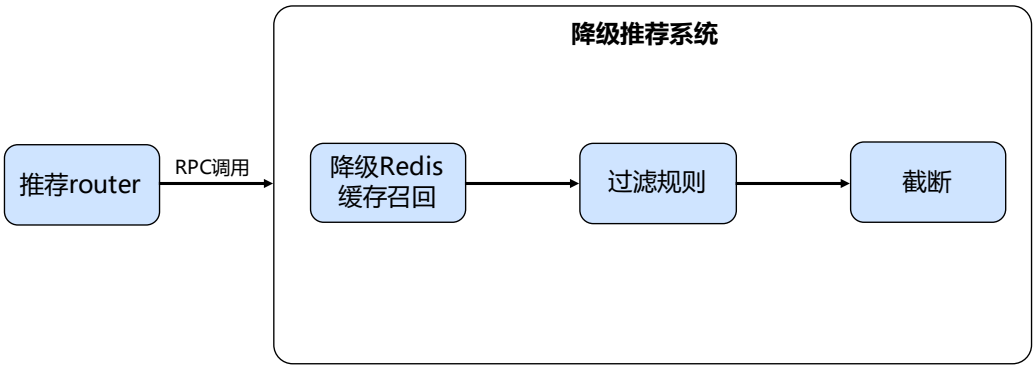


图 2.8: 降级推荐系统内部结构示意图。

最后，我们简要介绍降级推荐系统内部的实现结构。如图 2.8 所示，降级推荐系统本身通常不会重新执行完整的推荐流程，而是基于历史缓存结果进行轻量级处理，其主要包括以下三个步骤：

1. **降级 Redis 缓存召回**

   实时推荐系统在正常运行时，会将用户最近一次或最近若干次推荐请求对应的排序结果写入降级 Redis 缓存。当用户进入降级链路后，系统直接从缓存中召回对应推荐结果，从而避免执行完整推荐计算。

2. **过滤规则**

   由于缓存内容可能已经被用户浏览过，因此系统通常需要根据用户最近行为进行去重过滤。同时，还会执行风控过滤、内容安全过滤以及业务策略过滤等规则，以保证推荐结果的可用性与合规性。

3. **结果截断**

   由于 Redis 中通常缓存的候选数量大于单次推荐所需数量，因此系统会按照当前请求需要返回的 Top-K 结果进行截断。未被使用的部分结果则继续保留在缓存中，以供后续降级请求使用。

通过引入降级推荐系统，工业级推荐平台能够在流量高峰、资源紧张或系统异常等场景下维持服务连续性，从而在推荐效果、系统稳定性与算力成本之间实现更好的平衡。

## 2.5 作者侧推荐系统

在绝大多数工业级推荐系统中，推荐服务通常围绕用户请求展开：系统根据用户画像、历史行为以及当前上下文信息，从海量候选内容中筛选出最有可能引发用户兴趣的物品并进行推荐。这类以用户需求为中心构建的推荐架构，通常被称为**用户侧推荐系统（User-side Recommendation System）**。

然而，用户侧推荐系统在长期运行过程中往往会面临一个普遍存在的问题，即流量向头部内容过度集中。由于推荐模型的优化目标通常围绕点击率（CTR）、观看时长（Watch Time）、转化率（CVR）等用户价值指标展开，而历史表现较好的热门内容往往能够获得更高的预测分数，因此这些内容会持续获得更多曝光机会，并进一步积累新的用户反馈数据。与此同时，曝光较少的长尾内容由于缺乏足够的行为数据支撑，其预测效果往往不够理想，从而难以获得进一步曝光机会。长期迭代后，系统便会形成典型的**“马太效应（Matthew Effect）”**，即：

热门内容获得更多流量，而更多流量又进一步强化其热门地位。

这种现象最终导致推荐流量呈现明显的长尾分布特征。

如图 2.9 所示，从图 2.9(a) 可以看到，随着内容热度的提升，所获得的曝光流量呈现出显著的幂律增长趋势，少量头部内容占据了平台的大部分流量资源，而大量中尾部内容只能获得极少的曝光机会。从图 2.9(b) 可以进一步发现，虽然单个长尾内容获得的流量非常有限，但长尾区域聚集了平台中绝大多数内容供给。因此，从内容数量角度来看，长尾内容实际上构成了平台生态的重要组成部分。

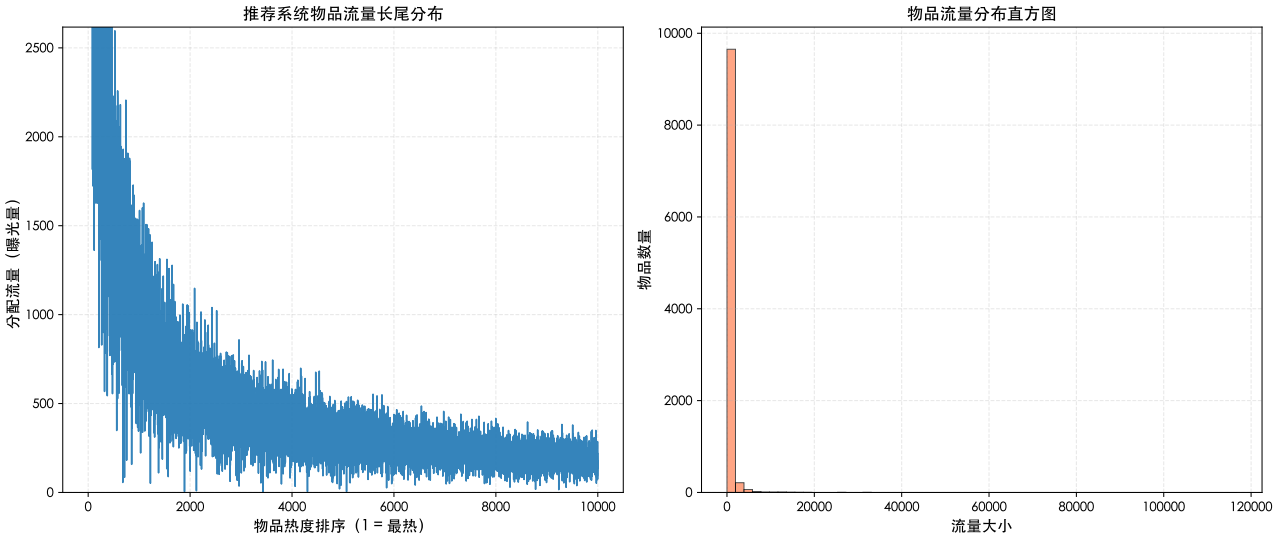


**图 2.9:** 用户侧推荐系统中存在的物品长尾分布问题。

从用户体验的角度来看，这种流量集中现象具有一定合理性。因为推荐系统的核心目标之一就是最大化用户价值，而热门内容通常更容易获得较高的点击率、观看时长以及互动率，因此将流量分配给这些内容往往能够带来更优的短期业务指标。

然而，如果从平台生态建设的视角来看，仅依赖用户侧推荐系统则可能带来新的问题。推荐系统本质上连接的是内容消费者与内容生产者，其承担的不仅仅是流量分发的职责，同时也影响着内容供给生态的长期演化。如果平台中的绝大部分流量持续流向头部创作者，而中腰部及新创作者长期无法获得成长机会，那么内容供给将逐渐趋于单一化，创作者活跃度也可能持续下降。对于一个平台的中腰部创作者而言，通常都会非常在意自己的流量好坏，以及自己发布的作品是否能够带来涨粉。如果自己发布的内容观看量很低，这就会打击创作者发布内容的积极性。长期来看，这将削弱平台内容生态的活力与创新能力，并最终反向影响用户体验。

因此，对于内容平台而言，推荐系统实际上需要同时优化两个目标：

- **用户价值（Consumer Value）**：提升用户满意度、观看时长、留存率等核心消费指标；
- **生产者价值（Producer Value）**：促进创作者成长，提高内容供给质量与生态繁荣度。

在工业界，通常将围绕创作者成长、内容供给激励以及生态建设开展的相关业务统称为**生产业务（Creator Growth Business）**。其核心目标并非单纯提高单次推荐的点击率，而是通过流量扶持、创作者激励、成长任务体系以及内容冷启动机制等方式，帮助中腰部创作者持续成长，从而提升平台整体内容供给能力。从某种意义上来说，用户与创作者构成了内容平台生态中的供需两侧：用户是内容的消费者（Consumer），而创作者则是内容的生产者（Producer）。如果缺乏持续增长的优质内容供给，即使短期内用户侧指标表现良好，平台也难以维持长期竞争力。

正是在这样的业务背景下，传统用户侧推荐系统开始暴露出其局限性。由于其优化目标主要围绕用户消费行为展开，因此很难直接解决创作者成长与内容供给相关的问题。为此，近年来工业界逐渐提出并探索一种与用户侧推荐系统相对应的推荐范式，即**作者侧推荐系统（Creator-side Recommendation System）**，也被称为**对偶推荐系统（Dual Recommendation System，DualRec）** [1]。

DualRec 的核心思想在于：

**定义 2.4**

首先从内容和创作者的视角出发，通过作者推荐系统找到当前在线同时对该内容感兴趣的用户。进而通过用户侧推荐系统与作者侧推荐系统相互结合与补充，将其纳入到一个完整的推荐框架之中，在提升用户体验的同时，也能促进内容生态的长期繁荣与可持续发展。

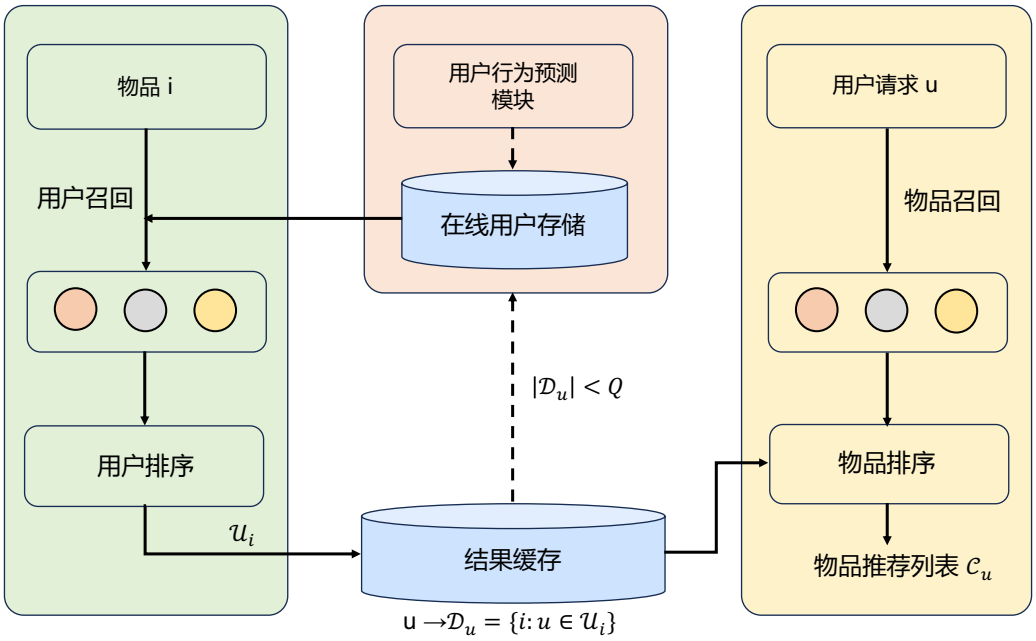


图 2.10: DualRec 系统架构图。

DualRec 的整体架构如图 2.10 所示。与传统推荐系统主要围绕“用户 → 物品”的匹配过程不同，DualRec 在用户侧推荐系统之外额外引入了一套作者侧推荐系统，从而形成用户侧与作者侧协同工作的对偶推荐架构。

在作者侧推荐系统中，推荐过程不再以用户作为起点，而是以内容或创作者作为起点展开。具体而言，对于一个待分发的物品（Item），系统首先从在线活跃用户集合中召回可能对该内容感兴趣的用户，并进一步通过排序模型评估用户与物品之间的匹配程度，最终得到该物品对应的一组目标用户集合。换句话说，传统推荐系统解决的是“给用户推荐什么内容”，而作者侧推荐系统解决的是 **“给内容寻找哪些潜在用户”**。

完成用户召回与排序后，系统会对推荐结果进行重组。对于每个用户 $u$，将所有通过作者侧推荐系统匹配到该用户的物品聚合形成候选集合：

$$D_u = \{item_1, item_2, \cdots, item_n\}, \tag{2.4}$$

其中，$D_u$ 表示作者侧推荐系统为用户 $u$ 生成的推荐物品集合。随后，系统以用户 ID 作为 Key，将对应的候选集合 $D_u$ 以 Key-Value 的形式写入 Redis 缓存，以供后续在线推荐阶段快速访问。

为了进一步提升作者侧推荐系统的计算效率，DualRec 通常会维护一个专门的**在线用户池（Online User Pool）**。一方面，对于作者侧推荐结果数量仍未达到用户日消费上限 $Q$ 的用户，即满足：

$$|D_u| < Q \tag{2.5}$$

，系统会将其加入在线用户池中，表示这些用户仍存在内容消费需求。另一方面，DualRec 还会引入一个用户活跃度预测服务，对用户未来一段时间内是否仍将保持在线状态进行预测。对于被预测为短时间内仍会活跃的用户，同样会被加入在线用户池。通过上述机制，作者侧推荐系统无需面向全量用户集合执行召回，而只需在在线用户池中寻找潜在目标用户，从而显著缩小召回空间，降低系统计算开销，并提高推荐结果的实时性。

作者侧推荐结果最终会与传统用户侧推荐系统进行融合，共同参与最终推荐列表的生成。当用户发起推荐

请求时，用户侧推荐系统首先按照常规流程完成召回与排序，同时从 Redis 中读取当前用户对应的作者侧推荐候选集合 $D_u$。随后，系统通过保量（Quota Protection）、Boost 加权、混排融合（Mix Ranking）等策略，将作者侧推荐结果与用户侧推荐结果进行联合排序，最终生成推荐给用户的物品列表。

从整体流程来看，用户侧推荐系统主要负责优化用户体验与流量效率，而作者侧推荐系统则更加关注内容分发效率与创作者成长。前者回答的是“用户想看什么”，后者回答的是“哪些内容值得被更多用户看到”。二者共同构成了 DualRec 的双向推荐机制，使推荐系统能够同时兼顾用户价值、创作者价值以及内容生态价值，在流量效率与生态繁荣之间取得更好的平衡。

## 2.6 系统日志与数据

推荐系统中的日志与数据是提升推荐模型效果的核心基础。从某种意义上说，推荐系统的竞争，本质上也是数据质量与数据利用能力的竞争。尽管我们习惯将 CTR 预估模型、排序模型乃至大模型推荐框架统称为“AI 模型”，但归根结底，它们本质上仍然是**高度依赖数据驱动的统计学习系统（Data-Driven Statistical Learning System）**。

这些模型本身并不具备先验知识，其对用户兴趣的理解、对内容价值的判断以及对用户—内容匹配关系的学习，全部建立在历史交互数据之上。无论是协同过滤、深度学习推荐模型，还是近年来兴起的大语言模型推荐框架，其能力的上限都在很大程度上受到训练数据质量的约束。如果缺乏高质量的用户行为数据，即使采用再复杂的模型结构，也难以获得理想的推荐效果。

与早期基于用户-物品评分矩阵（如 MovieLens 数据集）的推荐方法不同，现代工业级推荐系统所面对的数据环境更加复杂。

一方面，可利用的数据维度极其丰富。除了用户与物品之间的交互行为之外，系统还能够融合：

- **用户画像（User Profile）**：年龄、性别、地域、设备类型、活跃时间段等；
- **物品特征（Item Feature）**：类别、标签、Embedding、内容理解结果等；
- **上下文信息（Context Feature）**：时间、地点、网络环境、会话状态等；
- **行为反馈信息（Behavior Feedback）**：点击、观看时长、点赞、评论、分享、收藏、购买等。

另一方面，用户兴趣本身也具有明显的动态演化特征。用户可能因为热点事件而短期产生兴趣增强，也可能因为内容重复曝光而出现兴趣衰减，甚至随着年龄增长、工作变化或生活阶段迁移而发生长期兴趣转变。因此，推荐系统不仅需要学习用户当前喜欢什么，还需要持续跟踪用户兴趣的变化过程。

与此同时，工业场景中的数据规模通常达到 **TB 级甚至 PB 级**。每日新增日志可能达到数十亿乃至上百亿条记录。海量数据虽然为模型训练提供了充足的学习样本，但同时也带来了新的挑战：

- **数据质量问题**：日志中可能混杂客户端异常、埋点错误、网络抖动以及恶意刷量行为产生的噪声数据；
- **反馈稀疏问题**：绝大多数曝光并不会产生显式反馈，用户只会对极少数内容进行点赞、评论或购买；
- **反馈延迟问题**：部分目标行为（如转化、下单、长期留存）可能在曝光数小时甚至数天后才发生；
- **分布偏移问题**：训练数据来源于历史曝光，而历史曝光本身受到旧模型影响，从而形成曝光偏差（Exposure Bias）与选择偏差（Selection Bias）。

因此，如何从海量、高维、稀疏且含噪的数据中提取有效信号，并构建稳定、鲁棒且具备长期价值的推荐模型，已经成为现代推荐系统研究与工程实践中的核心课题。

### 2.6.1 流式数据与离线数据

从数据生产方式来看，工业界推荐系统中的数据通常以两种形式存在，即**实时数据流（Streaming Data）**与**离线批量数据（Batch Data）**。前文介绍推荐系统离线架构时提到，样本拼接服务会通过 Kafka 等消息队列实时消费用户请求日志与反馈日志，并持续生成训练样本。这类数据主要服务于近线（Nearline）与在线（Online）模型更新。需要说明的是，近线更新与在线更新并不完全相同。

- **近线更新（Nearline Update）**

通常指基于实时行为流持续更新用户特征或物品特征，例如最近点击序列、短期兴趣 Embedding、实时热度统计等，而模型参数本身并不发生变化。

- **在线更新（Online Learning）**
  则更进一步，模型参数会随着新反馈到来而实时学习与更新。例如在线 Logistic Regression、FTRL 等在线学习框架。

在实际工业系统中，近线特征更新远比在线参数更新更为常见。原因在于在线学习虽然能够获得更快的反馈响应速度，但同时也带来了模型稳定性、参数漂移以及线上风险控制等问题。

除了实时数据流之外，推荐系统还会定期将原始日志、用户特征快照、物品特征快照以及训练样本归档至离线数据仓库中。这些数据主要用于：特征分析与验证，离线模型训练，AB 实验评估，用户行为分析，问题排查与数据回溯等。虽然离线数据存在一定延迟，但其覆盖范围更加全面、数据质量更加可控，因此仍然是推荐模型迭代的重要基础。

### 2.6.2 Hive 与推荐系统数据仓库

在工业实践中，上述离线数据通常以 Hive 表的形式组织存储。Apache Hive 是构建在 Hadoop 生态之上的数据仓库系统，其核心作用是为海量数据提供类似 SQL 的查询能力。借助 Hive SQL（Hive Query Language, HQL），开发者无需编写复杂的 MapReduce 程序，便能够直接对存储于 HDFS 中的 PB 级数据进行查询与分析。对于推荐算法工程师而言，Hive SQL 几乎是一项必备技能，例如：分人群统计不同用户群体的行为反馈表现、分析不同创作者分层下的流量获取情况、分析 AB 实验结果等；几乎都需要通过 Hive SQL 完成。因此，Hive 可以被视为连接原始日志与推荐模型之间的重要桥梁。

### 2.6.3 推荐系统中的核心日志

在掌握 Hive SQL 这一基础工具之后，算法工程师便能够对推荐系统中的原始日志进行分析与特征构建。以短视频推荐场景为例，推荐系统中的日志通常围绕三个核心对象展开：**用户（User）、物品、用户与物品的交互（Interaction）**。因此，工业界推荐系统最常见的数据表通常可以归纳为三类：

- **用户侧基础信息日志（User Profile Log）**：通常记录用户的静态画像与设备属性，通常每日快照一次，用于刻画用户的长期兴趣背景与行为环境。字段包括：用户的用户 ID、设备 ID、年龄、性别、兴趣偏好、机型、手机系统、手机 App 版本、是否新用户、是否登录等。
- **物品侧基础信息日志（Item Metadata Log）**：描述物品内容及其元数据，由内容理解系统或创作者后台生成，通常在物品发布时写入，并随互动数据动态更新。字段包括：物品 ID、创作者用户 ID、创作者设备 ID、话题 tag、点赞数量、评论数量、分享数量、转发数量等。
- **用户—物品交互日志（Interaction Log）**：通常记录用户与视频在不同时间粒度下的行为，是推荐模型训练的核心信号来源。交互日志也可以分为两类：
  - **请求粒度的信息日志**：每次推荐请求生成一条记录，包含曝光上下文与即时反馈，用于构建实时训练样本。字段包括：用户与物品交互的请求 ID、用户 ID、设备 ID、物品 ID、创作者用户 ID、创作者设备 ID、观看时长、是否点赞、是否评论、是否分享、是否转发等；
  - **用户与物品天级交互日志**：按用户-物品对每日汇总行为统计，用于长期兴趣建模与特征回溯。字段通常包括：用户与物品交互的请求 ID、用户 ID、设备 ID、物品 ID、创作者用户 ID、创作者设备 ID、天级累计观看时长、天级累计点赞数量、天级累计评论数量、天级累计分享数量等。

#### 用户侧基础信息日志

表2.1 展示了用户侧基础信息日志的典型结构。该表通常以天为单位进行快照更新，用于构建用户的静态画像。其中，user_id 和 device_id 是跨会话行为归因的关键标识；age 与 gender 提供基础人口统计特征，常用于分群策略或特征交叉；interest_tags 由用户历史行为聚类或内容偏好模型生成，例如“# 美食,# 旅行”反映了用户对生

活类内容的长期兴趣；os 与 app_version 可用于分析不同设备或版本下的用户行为差异；而 is_new_user 字段则直接服务于冷启动逻辑，如第二行所示，用户 20240 为新用户（is_new_user = true），系统可能为其启用探索性推荐策略或默认兴趣兜底。

**表 2.1:** 用户侧基础信息日志示例

| user_id | device_id | age | gender | interest_tags | os | app_version | is_new_user |
|---|---|---|---|---|---|---|---|
| 10086 | d7a3b2c1 | 28 | female | # 美食,# 旅行 | Android 12 | v3.4.1 | false |
| 20240 | e9f8d7c6 | 19 | male | # 游戏,# 科技 | iOS 16 | v3.4.0 | true |
| 30571 | a1b2c3d4 | 35 | male | # 财经,# 新闻,# 汽车 | Android 13 | v3.4.2 | false |
| 40892 | f5e4d3c2 | 24 | female | # 美妆,# 穿搭,# 健身 | iOS 17 | v3.4.1 | false |
| 51203 | 9z8y7x6w | 42 | female | # 教育,# 亲子,# 家居 | Android 11 | v3.3.9 | true |
| 62314 | m2n3o4p5 | 17 | male | # 动漫,# 电竞,# 短视频 | iOS 15 | v3.4.0 | true |

## 物品侧基础信息日志

表2.2 描述了短视频的基础元数据。视频在发布时即写入初始信息，并随用户互动行为准实时更新计数类字段。video_id 作为内容唯一标识，是特征索引与样本拼接的核心键；author_id 与 author_device_id 用于关联创作者画像；duration_sec 不仅决定视频长度，还影响完播率等关键指标的计算；tags 通常由人工打标、自然语言处理或计算机视觉模型自动生成，如“# 萌宠,# 搞笑”可用于语义召回通道；而 like_cnt、comment_cnt、share_cnt 等互动统计数据，既是内容热度的直接体现，也可作为排序模型中的统计特征（例如经平滑处理后的分享率）。例如，视频 v12345 虽仅 45 秒，但已积累上万点赞，表明其具有较高用户吸引力。

**表 2.2:** 短视频基础信息日志示例

| video_id | author_id | author_device_id | duration_sec | tags | like_cnt | comment_cnt | share_cnt |
|---|---|---|---|---|---|---|---|
| v12345 | 70349 | z5f8c5k0 | 45 | # 萌宠,# 搞笑 | 12500 | 842 | 310 |
| v67890 | 81426 | q2w9e7r4 | 28 | # 旅行,# 海岛 | 3200 | 156 | 98 |
| v24680 | 93571 | m3n8b2v6 | 62 | # 知识,# 科普 | 8900 | 1203 | 742 |
| v13579 | 64802 | t7y4u9i1 | 18 | # 美妆,# 教程 | 21000 | 2450 | 1830 |
| v97531 | 55128 | j6h3g9f2 | 55 | # 剧情,# 情感 | 45600 | 3890 | 2100 |
| v86420 | 47293 | x8c4v6b1 | 12 | # 音乐,# 卡点 | 6700 | 320 | 410 |

## 用户—物品交互日志

表2.3 展示了请求粒度的用户-短视频交互日志，这是推荐系统中最核心的“曝光-反馈”数据。每条记录对应一次推荐请求，包含完整的上下文与多维反馈信号。req_id 唯一标识一次请求，是样本拼接的基础；watch_sec 表示实际观看时长，结合视频总时长可衍生出多种行为标签：例如，若观看时长小于 3 秒，则 is_short_watch = true，通常视为无效曝光（负样本）；若大于 6 秒，则 is_valid_play = true，作为弱正样本信号；若观看时长等于或接近视频总时长，则 is_complete = true，反映内容具有强吸引力。此外，is_like、is_comment、is_share 等显式反馈行为共同构成多目标学习的标签体系。例如，第一行中用户 10086 观看视频 v12345 总共 42 秒（未完播），但进行了点赞和分享，表明其对内容高度认可；第二行中用户 20240 完整观看了 28 秒的视频 v67890 并评论，说明内容与其兴趣高度匹配。这些细粒度信号为精排模型的多任务学习提供了丰富监督信息。

在实际工程中，上述三类日志通常通过 Hive SQL 的 JOIN 操作进行关联，从而构建推荐模型所需要的训练样本：$(user, item, context, label)$。其中，User 对应用户特征，Item 对应物品特征，Context 对应上下文特征，Label 对应用户反馈标签。这也是现代推荐模型训练数据的标准组织形式。

表 2.3: 用户-短视频交互日志示例（请求粒度）

| req_id | user_id | device_id | video_id | author_id | author_device_id | watch_sec | is_like | is_comment |
|---|---|---|---|---|---|---|---|---|
| r98765 | 10086 | d7a3b2c1 | v12345 | 70349 | z5f8c5k0 | 42 | true | false |
| r87654 | 20240 | e9f8d7c6 | v67890 | 81426 | q2w9e7r4 | 28 | false | true |
| r76543 | 30571 | a1b2c3d4 | v24680 | 93571 | m3n8b2v6 | 2 | false | false |
| r65432 | 40892 | f5e4d3c2 | v13579 | 64802 | t7y4u9i1 | 18 | true | true |
| r54321 | 51203 | 9z8y7x6w | v97531 | 55128 | j6h3g9f2 | 55 | true | false |
| r43210 | 62314 | m2n3o4p5 | v86420 | 47293 | x8c4v6b1 | 5 | false | false |

### 2.6.4 大模型驱动的内容理解数据

然而，仅依赖传统元数据仍然存在明显局限。例如：视频时长，视频标签，作者信息，点赞数这些字段虽然能够描述内容的一部分属性，但往往难以刻画内容真正的语义信息，因而推荐系统也无法真正理解这个物品的语义知识究竟是什么。随着大语言模型（LLM）与多模态大模型（VLM）的快速发展，推荐系统开始逐渐从“行为驱动推荐”向“行为 + 语义双驱动推荐”演进。

在短视频场景中，LLM 可以分析 ASR 文本，LLM 可以总结评论区观点，LLM 可以自动生成主题标签，VLM 可以理解视觉内容，VLM 可以生成统一的多模态 Embedding。这些能力使推荐系统能够从更深层次理解内容本身。因此，工业界通常会构建一张由大模型增强的内容元数据表。

为支持上述能力的工程落地，我们构建了一张 LLM/VLM 增强的短视频元数据表（见表 2.4）。该表以 video_id 为主键，并新增由模型生成的 asr_summary、topic、category、comment_summary 与 visual_emb 字段。其中，asr_summary 是 LLM 对视频语音转文字内容生成的简洁摘要，topic 表示核心讨论主题，category 为模型判定的垂类标签，comment_summary 是对高赞评论的语义聚合结果，visual_emb 则为 VLM 输出的稠密向量（通常为 768 维），在表中仅展示前几位作为示意。该表可与用户交互日志、用户画像等表通过 video_id 高效关联，显著增强训练样本中“item”与“context”的语义丰富度。

表 2.4: LLM/VLM 增强的短视频元数据表示例

| video_id | asr_summary | topic | category | comment_summary | visual_emb |
|---|---|---|---|---|---|
| v12345 | 一只橘猫被逗猫棒逗得疯狂扑跳，全程搞笑不断。 | 萌宠互动 | 宠物 | “猫咪反应太可爱了！笑到肚子疼” | [0.82, -0.31, 0.45, …] |
| v67890 | 探访马尔代夫私人岛屿，展示日落海滩与水上别墅。 | 海岛旅行 | 旅行 | “想去这个地方！摄影太美了” | [-0.12, 0.67, -0.23, …] |
| v24680 | 用生活实例解释牛顿三大定律，通俗易懂。 | 物理科普 | 知识 | “讲得通俗易懂，期待下一期” | [0.55, 0.28, 0.91, …] |
| v13579 | 演示新手如何画出自然日常眼妆，步骤清晰。 | 美妆教程 | 美妆 | “步骤清晰，已收藏反复看” | [0.33, -0.44, 0.76, …] |
| v97531 | 讲述单亲父亲独自抚养女儿的温情故事。 | 亲情 | 剧情 | “看哭了，真实得让人心疼” | [-0.61, 0.19, -0.52, …] |
| v86420 | 卡点舞蹈配合热门 BGM，节奏感极强。 | 舞蹈卡点 | 音乐 | “节奏感绝了！BGM 求链接” | [0.70, 0.88, -0.15, …] |

通过将此类由 LLM 与 VLM 生成的语义增强数据与用户行为日志深度融合，推荐系统的能力边界得以显著拓展：不仅能够基于历史交互“知道用户喜欢什么”，更能借助深层次的内容理解“洞察内容为何被喜欢”。这种从行为信号驱动向语义理解驱动的范式跃迁，使得推荐系统在捕捉用户兴趣、识别内容价值以及建模用户-内容匹配关系等方面更加精准、鲁棒且具备可解释性。例如，当模型理解到某视频被喜爱是因为其“通俗易懂的物理讲解”而非仅因“高完播率”，便能在相似语义场景下实现更泛化、更合理的推荐。上述由大模型赋能的内容理解能力，已逐步成为新一代智能推荐系统的核心基础设施。本文后续章节将系统性地展开讨论：包括 LLM 如何处理

ASR 文本与用户评论以构建结构化语义标签，VLM 如何生成高质量的多模态表征并应用于召回与排序阶段，以及如何将这些大模型输出有效融入多任务学习框架，实现语义信号与行为信号的协同优化。

综上所述，日志与数据构成了推荐系统的基础设施层。从用户行为采集、离线数据仓库建设，到训练样本构建，再到大模型驱动的内容理解，整个数据体系持续为推荐模型提供学习所需的“燃料”。随着 LLM 与 VLM 技术的发展，推荐系统正在逐渐突破传统行为建模的边界，从“根据用户过去点击过什么进行推荐”，演进到“理解用户为什么喜欢某类内容以及内容本身表达了什么”。这种从**行为信号驱动（Behavior-driven）**向**语义理解驱动（Semantic-driven）**的演进，正在成为下一代智能推荐系统的重要发展方向。后续章节中，我们将进一步介绍大语言模型与多模态大模型如何参与内容理解、用户建模以及召回与排序优化，并探讨其在推荐系统中的实际落地方式。

## 2.7 本章小结

本章在上一章推荐系统概述的基础上，进一步从系统工程的视角介绍了工业级推荐系统的整体架构。首先，我们从多业务混合推荐系统架构出发，介绍了推荐系统在实际业务中的复杂组织形式，并分析了混排模块在多业务流量分配与结果融合中的作用机制。随后，我们聚焦于单一业务场景下的经典级联式推荐架构，系统梳理了召回、粗排、精排以及重排等核心模块之间的协同关系，帮助读者建立完整的推荐链路认知。

在此基础上，本章进一步介绍了推荐系统的离线架构与在线架构。其中，离线架构主要关注用户行为数据的采集、样本构建、特征生成、索引维护以及模型训练等流程，重点讨论了用户侧与物品侧特征的存储方式及其更新机制；在线架构则更加关注推荐服务的低延迟、高并发、高可用等工程问题，以及实时推荐过程中对于稳定性、安全性和资源利用效率的要求。

随后，我们介绍了工业界常见的降级推荐系统与作者侧推荐系统（DualRec）。其中，降级推荐系统通过动态算力分配与缓存机制保障推荐服务在高并发场景下的稳定运行；作者侧推荐系统则从内容生产与创作者生态的角度出发，利用推荐系统的对偶性实现用户价值与内容生态价值的协同优化。

最后，我们介绍了推荐系统中的日志与数据体系，包括用户侧基础信息日志、物品侧基础信息日志、用户—物品交互日志以及由大语言模型和多模态大模型生成的内容理解数据等。通过这些数据资产，推荐系统得以持续学习用户兴趣、理解内容语义，并不断优化推荐效果。

通过本章的学习，读者不仅能够理解推荐系统各个核心模块的功能与职责，还能够从整体视角认识工业级推荐系统的数据流、服务流以及模型流是如何协同运作的，从而为后续深入学习推荐模型、排序算法以及大模型推荐技术奠定坚实的系统基础。

## 2.8 参考文献

[1] Xiaoshuang Chen et al. “Creator-Side Recommender System: Challenges, Designs, and Applications”. In: *Companion Proceedings of the ACM on Web Conference 2025*. 2025, pp. 162–170.

[2] Biye Jiang et al. “Dcaf: A dynamic computation allocation framework for online serving system”. In: *arXiv preprint arXiv:2006.09684* (2020).