本科生实验报告

实验课程: 计算机体系结构

授课老师: 沈明华

专业名称: 计算机科学与技术

学生姓名: 庄钰堃

学生学号: 21307420

实验时间: 20240101

复现论文: End-to-End Deep Learning Proactive Content Caching Framework

论文作者: Eslam Mohamed Bakr Hamza Ben-Ammar Hesham M. Eraqi Sherif G. Aly Tamer Elbatt Yacine Ghamri-Doudane

论文发表: GLOBECOM 2022 - 2022 IEEE Global Communications Conference

Section 1 研究背景

本篇论文的主要解决问题是针对缓存的优化，通过借鉴了推荐系统里面协同过滤的启发，设计了一种端到端的缓存框架。

在过去几年中，许多互联网服务和应用程序逐渐迁移到移动网络。这直接导致了移动流量的急剧上升，因此如何在这种情况下保持用户的使用体验是正在面临的挑战。内容缓存技术便是面对挑战提出的一项技术——旨在通过在本地基站、服务器或移动和边缘设备上存储用户可能频繁请求的流行内容来克服此类挑战和提升用户体验。

采用短距离无线通信技术使这些移动的物联网设备能够形成自组织网络，该网络依赖于附近多个具有一定存储能力的物联网设备上的内容缓存，以实现可靠的内容传递。此外，对于内容分发网络 **CDNs:** 依靠在多个地理位置存储内容的缓存版本来处理大量的需求增长，提高网络性能和用户的 **QoE**，也对缓存及其算法有着较高需求。

深度强化学习是一项热门的技术，也因此被广泛应用于网络内容缓存。在文章：***Deep reinforcement learning for mobile edge caching: Review, new features, and open issues*** 中，为了激发对边缘缓存使用深度学习的动机，作者们进行了移动边缘缓存关键挑战与独特 **DRL** 方面之间的系统映射。在这方面，现有的基于 **DRL** 的缓存方法可以分为两类：（**i**）间接缓存参数学习器；和（**ii**）直接缓存学习器。前一组估计特定缓存相关参数的值和演变，如内容受欢迎程度和缓存过期时间。而后者，**DRL** 被直接用于学习最佳缓存策略。

由于在前几年 **LSTM** 的整体成功，它也被引入使用到内容缓存

方案中。一种基于联邦学习的主动内容缓存（FPCC）方案，属于上述第二类，他们的提议旨在提供高效的缓存方案，并同时通过避免为训练目的集中用户数据来保护用户的隐私，采用联邦学习（FL）方法。

此外，一个基于增量学习的边缘网络和 CDN 中数据缓存的框架被提出。在这个工作中，不是每次系统动态变化都重新构建一个新的缓存模型，而是保留了从先前模型中获得的有价值的知识。同一模型在逐渐改进的同时，更快更有效地适应动态工作负载。

还有的作者提出了 LFO（从 OPT 学习），其中使用梯度提升决策树混合，从 OPT 和请求的特征中学习，从而做出缓存决策。

总结一下对于目前对于内容缓存，主要问题是在给定有限的缓存容量的情况下，缓存或删除哪个对象以及何时缓存。由于缓存单元的存储限制，预测未来的内容请求模式并使用它来主动缓存最受欢迎的项目(即在需求之前缓存)是至关重要的。而目前的一些关于缓存的主流算法并不能预测项目未来的流行程度。

Section 2 实验概述

本论文作者的出发点便是在传统 cache 算法中引入“预测”的思想：借鉴了 recommend system 里的协同过滤思想，并结合了神经网络模型（NCF：Neural Collaborative Filtering），并改进提出了一种新的方案。

通过阅读我发现，作者的 proactive cache 的思想来自于 2014 年

的一篇文章：**Living on the Edge: The Role of Proactive Caching in 5G Wireless Networks.** 主动缓存我认为可以理解为一种系统通过预测的方式，预测用户将会感兴趣、热度高的内容，将这类信息保留在距离用户较近的服务器的 **cache** 当中，这样，那些经常被调用的信息因为被保留在了较近的服务器，所以可以更快被用户得到，由此可以提升 **QoS/E: Quality of Service/Experience,**进而提升整个 **CDNs(**内容分发网络**)**的性能。并且由于这种预测的思想，刚好契合近年来比较火热的推荐系统算法，尤其是再结合了神经网络模型后性能优越的 **NCF** 模型，也正因为这两个主要的出发点，作者提出了本文这个创新的缓存模型。

本文作者的工作主要如下：提出了“主动内容缓存的端到端模型”，并且作者提出了两种版本，其中第二版经过了第一版模型的改进，被称作：**Deep Learning Proactive Content Caching Framework**（**DLC**）

该模型通过深度神经网络构建每个用户内容项之间的概率分布来执行缓存任务，该网络对用户和项特征之间的交互进行建模。与原始的 **NCF** 模型不同，本框架将 **NCF** 模型扩展为直接学习主动缓存决策。此外，该框架还支持分布式和集中式缓存方案。最后，在一些真实公共数据集上，数据表明，它的性能明显优于最先进的主动缓存方案。

Section 3 实验原理

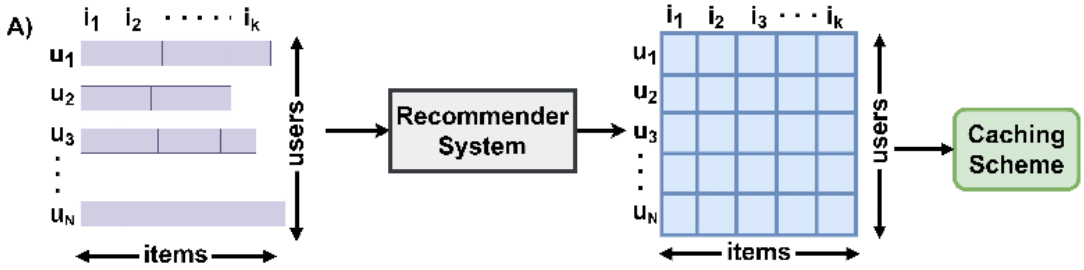
作者首先提出了“两阶段-缓存”算法，这是一种借用了“推荐系统”中协同过滤算法的方案（通过不同的策略计算 item-goods 的关联矩阵以达到打分（rating）结果，最终完成预测）。

此外，作者思考了这种方案的弊端，提出了新的改进点，引出了最终本文的 DLC 模型。

（接下来的原理分析按两部分进行分别介绍）

PART 1

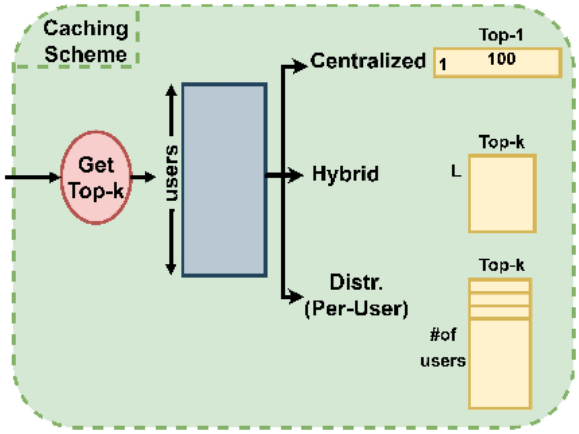
首先我们关注一下作者的第一个提出的模型“二阶段-缓存”模型。



可以分为两部分来解释该算法：

1）推荐系统函数 f(M)，其中 f 是非线性函数，M 是用户-项目交互矩阵，作者认为 f(M)的结果能够捕捉用户内部相关性、内容内部相关性和用户-内容之间的交叉相关性。（参考上图）

2）缓存方案：负责将矩阵编码为缓存矩阵 H∈(N×Kh)，其中 N 是用户数量，K 是项目的数量，Kh 是可配置的参数，表示每个用户偏好的首选项目的数量。简单来说就是第一问得到的 f(M)矩阵，按照每个用户最是配的 top-Kh 个内容进行提取，最终得到的矩阵的维度是 N x Kh。其中关于如何决定哪些用来缓存，作者也提出了几种构思，如下：



1）The Centralized approach：根据内容的频率对其进行排名，然后基于预测矩阵 **M** 获取所有用户的前 **k** 个内容。因此，缓存矩阵 **H ∈（1 x Kh）**,这在内存占用方面是最高效的。

2）The Per-User approach：根据其预测分数对内容进行排序，并存储个性化的缓存版本，其中每个用户都有其缓存矩阵 **H∈ (1 x Kh)**。这个设置模仿了一个“分布式”设置，其中有一个中央节点处理数据，但最终为每个用户存储一个定制的缓存矩阵。如果我们在中央节点连接每个用户的整个定制矩阵，缓存矩阵将成为 **H ∈(N x Kh)**。

3）The Hybrid approach 混合方法是效率和效果之间的一种可能的折衷，其中缓存矩阵 **H** 既不像按用户方式那样庞大，也不像中心化方法那样简单。在这个设置中，我们将用户分成 **L** 组，对于每个组，应用中心化方案，得到 **H∈ (1 x Kh)** 然后通过连接所有组，最终的缓存矩阵将是 **H ∈(L x Kh)**。

为了使得该模型框架更为灵活、适应不同的场景，作者实现了 **hybrid** 方法，那么还会面临的一个问题就是应该采用多少 **L** 能够得到较为理想的效果。

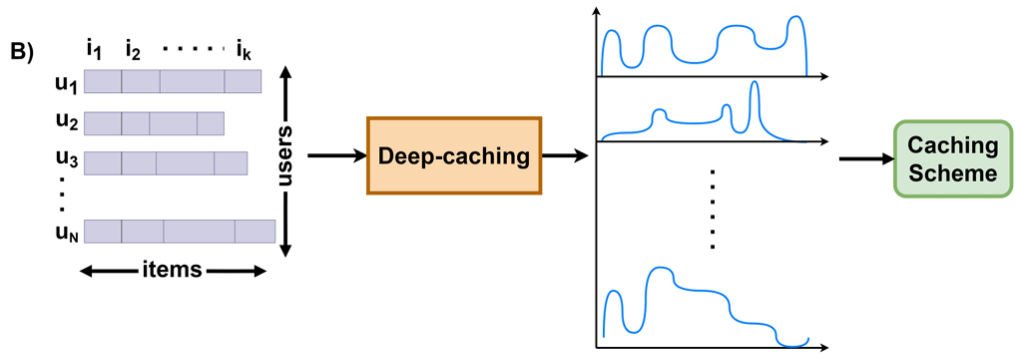
L \Cache Size	Rating Threshold=0				
	100	200	300	400	500
1	21.9	35.7	45.7	53.9	60.4
10	22.6	36.5	46.5	54.7	60.8
20	23.4	37.2	47.4	55.1	61.2

值得注意的是当 L 为 1 时候，就相当于是采用了上文说的 centralized 的方法。

我们观察可发现：作者的消融实验表明，在 1 至 20 间，L 的大小越大性能表现越好。这是合理的，因为 L 代表的其实是用户的聚类，最终得到的缓存矩阵 H 的行数就是 L,它代表了 L 类典型的用户，而对于庞大的数据集来说，比如本文使用的“MovieLens Data Set”，海量的用户需要有一定类别才能够基本涵盖一些典型的用户，我认为本实验验证到 L=20 也是合理的，毕竟人格测试 mbti 也才 12 种左右。也正因此，本实验的设置都是将 L 置为 20

PART 2

作者在上述模型的基础上，据推荐系统缓存内容，提出了一个端到端的框架，称为 DLC。



（原理如图所示）

动机：在之前的两阶段缓存模型中，网络架构和训练目标被设计来解决不同的任务，即推荐系统，其中神经网络被训练来解决二元分类问题，给定用户-项目交互。因此，使用神经网络的任务是回答交互是否存在？

既然如此，是否可以设计一个直接解决内容缓存问题而不使用推荐系统作为中间阶段的框架？

有了此想法，作者提出了一个端到端框架，通过利用神经网络架构直接解决内容缓存任务，学习每个用户基于偏好选择的前 **k** 个项目。

细节如下：对于给定的单个用户，目标是生成横跨整个 **K** 项的概率分布。网络通过 **SoftMax** 运算输出的条件概率可以估计为：

$$P(I_k = i|u_n) = \frac{exp(w_i^T u_n)}{\sum_j^C exp(w_j^T u_n)},$$

（其中[w1, ..., wC]是最后一个全连接层，d 是嵌入的维度，C 是类的数量。在端到端范式中，C = K，其中 K 是项目的数量）

通过制定训练目标以生成特定用户与所有 **item** 之间的似然度量，使模型能够捕获用户和项目之间的内部相关性，从而能够从矩阵 **M** 中学习用户与给定样本交互中未提及的其他项目之间的关系。

在推理过程中，作者将每个用户的特征馈送到网络中，也因此网络生成此用户在数据中表示的所有项目上的偏好分布。因此，这样的系统的复杂性仅为 **O(N)**，其中 **N** 是用户数量，使得与两阶段方法相比，随 **N** 的线性增长更加高效。

Section 4 实验步骤与复现结果

1）数据集的准备：

论文里绘图采用的数据都是使用了：**MovieLens**。这是一个电影评分数据集，被广泛用于评估协同过滤算法，包含一百万个评分的版本，每个用户至少有 **20** 个评分。这些数据是由将近 **6000** 个用户在 **1995** 年 **1** 月 **9** 日至 **2015** 年 **3** 月 **31** 日期间创建的，当时他们被要求对一组电影进行评分。因此，数据中可用的电影数量约为 **3700** 部。

使用该数据集在本实验会有一些需要面临的问题，可能会影响模型的训练效果，主要分为两类：

- **时间性问题：**数据提供了每个交互的时间戳信息。但是这些信息是具有误导性的，因为它们并不表示用户随时间的观点变化，而是用户一次性对所有电影进行评估。因此，如果我们根据时间戳信息将数据分割成不同的时间段，将导致不平衡的时间段，每个时间段都包含一组唯一的用户，在后续时间段中不会再次出现。为了解决上述的时间问题，处理方案是故意对数据集进行了排序，以确保每个时间段都包含所有用户的表示。
- **噪音信号：**每个用户只对有限数量的电影进行评分。评分范围从一到五，其中一表示用户完全不喜欢，五表示用户非常喜欢该电影。未评分的电影是具有误导性的信号，因为它们的出现可能由于各种可能性。对此，处理方案是对未评分的内容进行采用，用作负样本。

2）实验环境：

主要依赖项：

*** python==3.7**

*** pandas==0.24.2**

*** numpy==1.16.2**

*** pytorch==1.7.1**

*** gensim==3.7.1**

复现设备： CPU:R7-5800H GPU:RTX3060

搭建与配置：

可采用 **anaconda** 进行包管理，新建一个虚拟环境为 **DLC**，并在环境中配置本次实验所需要的依赖性和包，详细的配置信息在 **content_caching.yml** 文件中都有提供，比较简便的方式是直接在 **pycharm** 中打开该项目，然后换上虚拟环境 **DLC** 作为解释器，然后直接对于那些未安装的头文件安装，随装随用。

参数说明：

*** time_sorting** -->如果激活了该选项，我们将根据时间戳对数据进行排序，否则将进行分布以确保所有用户在每个时间间隔中都有表示。

*** rating_th** -->根据排序对数据进行筛选。例如，删除评分低于 **3** 的项目，只考虑从 **4** 开始的正面评分。如果将其设置为零，则表示不进行任何筛选。

*** window_split** --> 如果设置了该选项，数据将被分割成窗口。否则，如果设置为零，则使用所有的数据。请注意，此数字表示千位数。

*** E2E** --> 如果激活了该选项，这意味着我们将训练神经网络模型

直接预测前 k 个项目，而不是分为两个阶段。

*** modes --> 选择训练模式或者是测试模型**

实验设置：

所有模型都从零开始进行了 50 个 epochs 的训练，使用了权重初始化策略，并将初始学习率设置为 0.1，在每 30 个 epochs 时按 10 的倍数递减。使用了带有 1e-4 权重衰减、动量为 0.9 和小批量大小为 32 的随机梯度下降（SGD）。

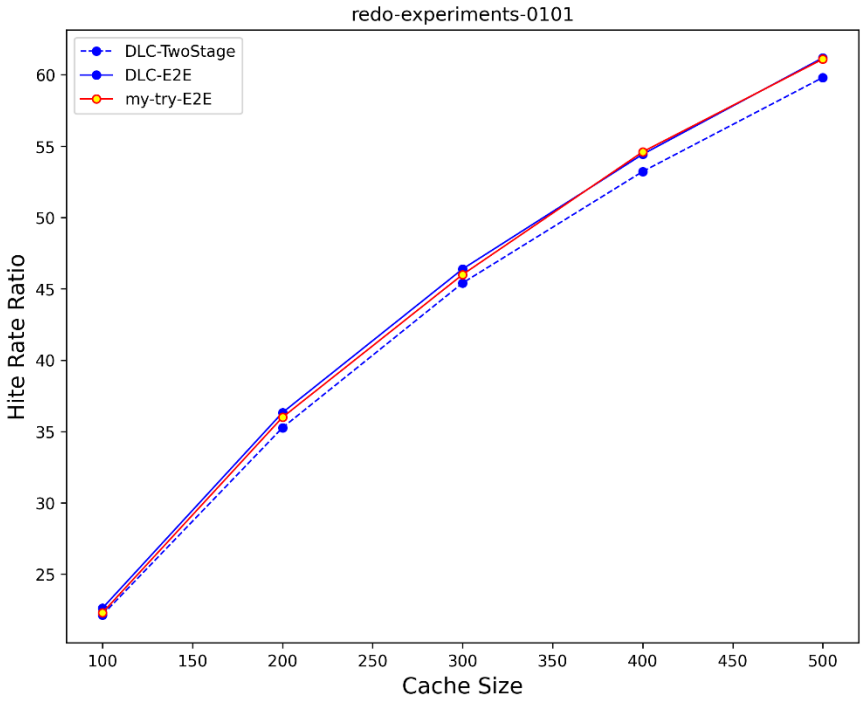
运行案例：

python main.py --batch_size=256 --lr=0.001 --factor_num=16

实验复现：

1）最终结果展示：

首先是我在本地复现的最终结果：



上图的红色线是我按照论文中采用的配置进行的，具体来说就是指定窗口大小 **WS**、组数 **L** 和评分阈值分别设置为 **500K**、**1** 和 **4**,可以发现效果非常不错，基本接近了本实验中的预期。

而对于本文首先提出的两阶段-缓存模型，我也进行了结果复现，运行结果如下：

9766/9766 [00:57<00:00, 169.70it/s]

The time elapse of epoch 005 is: 00: 01: 03

HR: 0.588 NDCG: 0.340

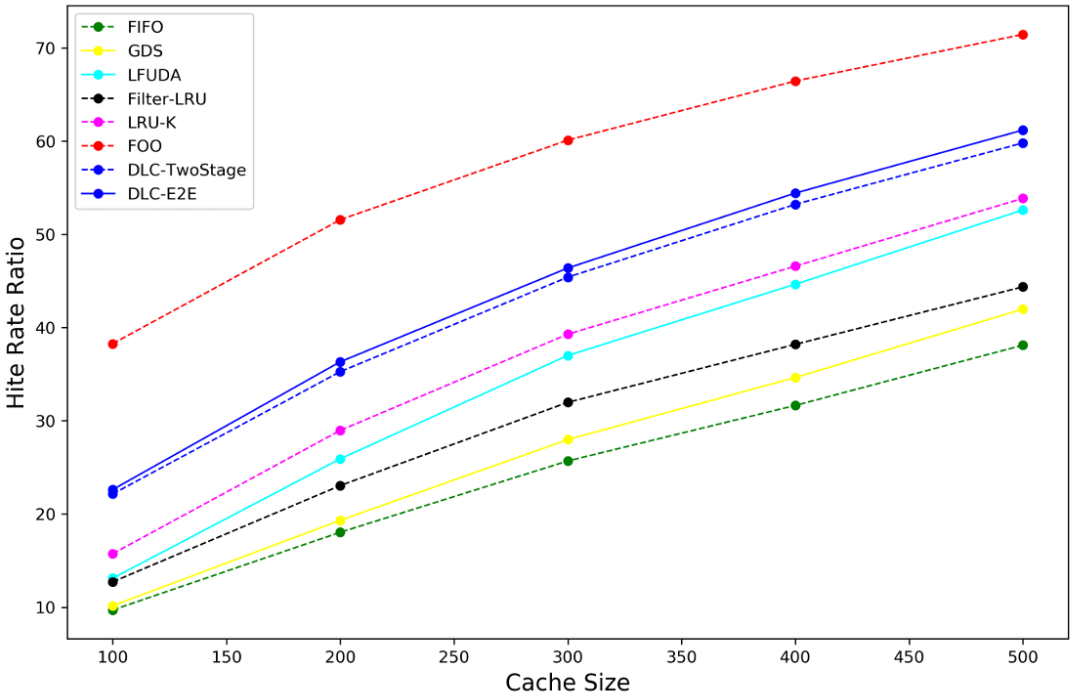
Epoch # 6

100%|

结果如上图，我截取了运行最佳的 **epoch 6** 进行展示。

通过对比上图的 **cache=500** 的蓝色曲线，可以发现复现的结果与论文中的结果几乎一致。

因此可以发现在本数据集中，作者所提出的两个模型——两阶段模型和 **DLC** 模型都达到了预期，并且表现好于其他现有的 **cache** 算法跑出来的模型，对比效果图如下：

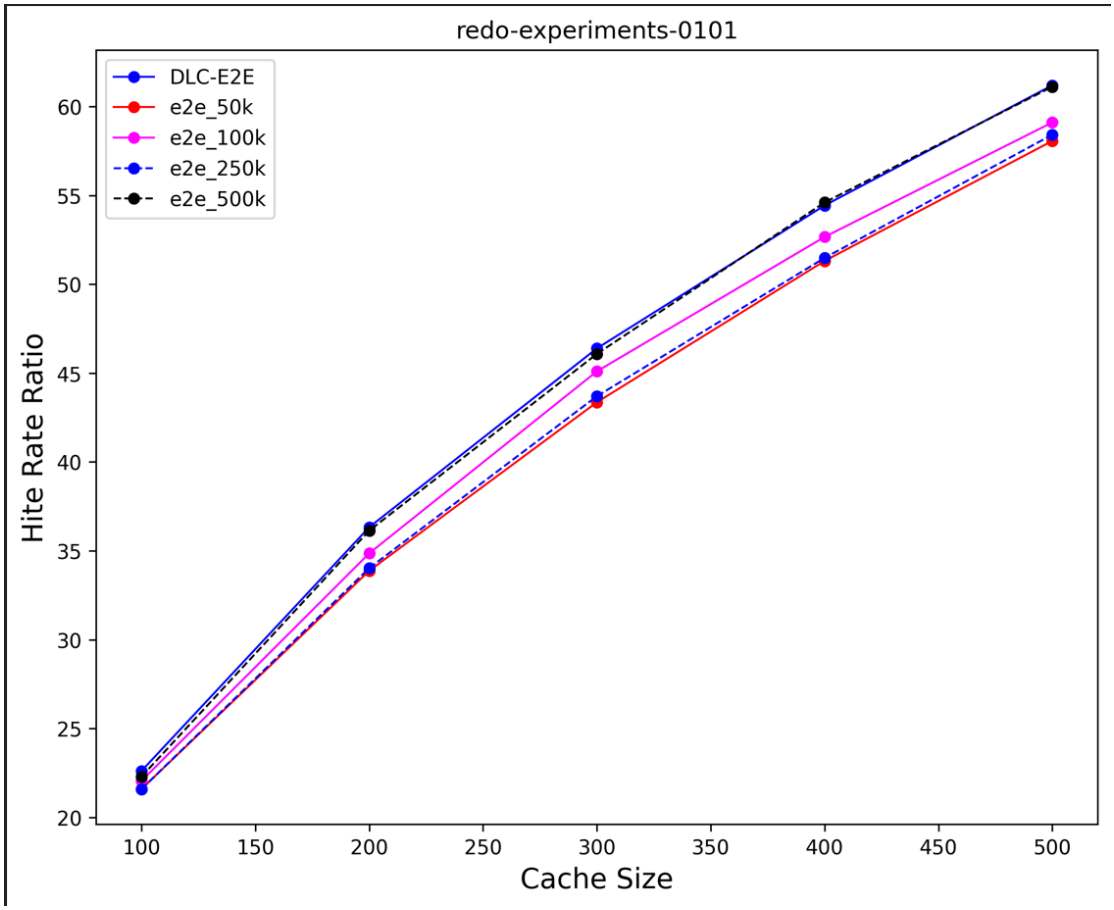


我们初步验证图上的蓝色的曲线（也就是作者提出的这两种模型）与实际运行结果相一致，并且不出意外的话，可以认为本实验的模型在本数据集上的表现优秀，可以对缓存有所提升。

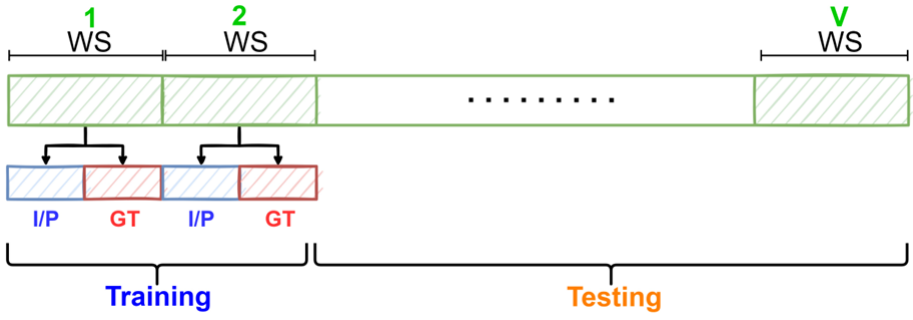
（2）消融实验复现：

接着我对于作者在文中提出的几个消融实验进行了复现，绘图结果如下：（均采用收敛后的最佳数据进行绘图）

2.1 关于 WS 滑窗的消融实验：



上图中我主要修改了滑动窗口的大小，分别从 **50k** 测试到了 **500k**，可以发现，不出意外的情况下，**500k** 的效果是会比较好的，并且随着缓存的增加，较大的窗口大小对于性能的提升也更显著。

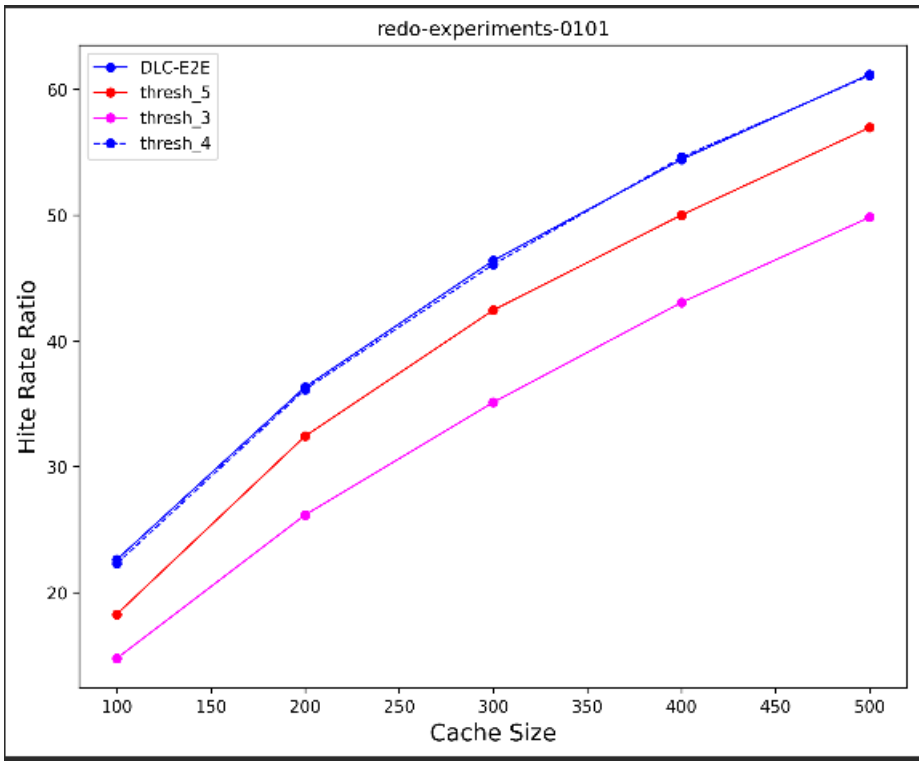


滑窗的原理展示如图，我们发现，当窗口越大，意味着参与的训练数据以及 **label** 就越多，那么显然训练效果会更好，这也与实验结果相匹配。然而现实中，由于我们处理的是用户缓存，这并不能做到

在短时间内积累非常多的数据(这可能会导致模型更新慢而减小性能，并且窗口过大也可能导致模型过拟合而效果不佳)，因此探究并找到一个合适的 WS 是有现实意义的。我认为从结果出发，可以折中地采取 250k 作为一个合适的取值。

2.2 关于评分阈值的选择：

结果展示如下：



从图中可以发现过高的阈值和过低的阈值都不能产生好的结果，当阈值取得 4 的时候表现明显最优，并且作者的模型也都是采用设置阈值为 4 来训练，我们可以发现黑色的虚线就是作者训练的模型效果，与蓝色的线几乎重合。

Section 5 实验思考与总结

一、绘图上出现错误

本次实验有一个作图上的错误，我在实验的时候发现，作者在 **TABLE 3** 提到的数据可能存在问题：

E2E	50K	**21.06**	**34.77**	**43.72**	**51.86**	**55.94**
	100K	**21.98**	**35.93**	**44.26**	**52.93**	**59.51**
	250K	**22.62**	**36.33**	**46.39**	**54.43**	**61.19**

TABLE III

上图作者在滑窗为 **250K** 的时候得到的这一些列结果经过我的实测，其实是没法达到的。

如果采用的滑窗的大小为 **250K**，那么：

最佳收敛结果大概如下：

```
The time elapse of epoch 003 is: 00: 00: 35
HR: 0.492       NDCG: 0.279
Acc at  100 :  0.20296297217317782
Acc at  200 :  0.3329737895705369
Acc at  300 :  0.43456096287270285
Acc at  400 :  0.5112432546688882
Acc at  500 :  0.5791472906771442
 Epoch #  4
```

而使用滑窗为 **500K** 的话，结果如下：

```
The time elapse of epoch 006 is: 00: 01: 31
HR: 0.603       NDCG: 0.360
Acc at  100 :  0.2230001115697869
Acc at  200 :  0.3613605935512663
Acc at  300 :  0.46082505857413814
Acc at  400 :  0.5459946446502287
Acc at  500 :  0.6110816690840121
 Epoch #  7
```

可以明显发现二者有明显区别，并且当 **WS** 为 **500K** 时，得出的

结果才和作者图示中一样。

而且，我在刚开始运行代码的时候发现，默认采用的滑动窗口大小也是设置在了 **500k**，这就说明，比较大的可能是这个图画错了。

二、数据集使用较少

因为本文的数据集使用的是推荐系统任务的数据集来作缓存的测试，我认为实验的验证还可以更全面一些 **Netflix Prize**（由 Netflix 提供的大规模推荐系统竞赛数据集。它包含了数百万用户对电影的评分数据）、**Amazon Product Data**（亚马逊提供了大量的产品数据，其中包括用户对产品的评分和评论。这个数据集可以用于构建推荐系统）、**Last.fm**（音乐流媒体平台，它提供了用户对音乐的听歌记录和标签数据，可用于构建音乐推荐系统）进行更多的验证，此外我认为还存在一个问题是，电影数据集是否能够准确的评测出缓存性能？我认为在这个方面也有待考虑，因为作者得出的结论确实是发现：常规的缓存算法在电影评分预测这个方面表现不好，但这样的评测是否有失偏驳不够有说服力呢？因为缓存面临的 **CDNs** 处理的数据比起这些复杂得多，有可能这个新的算法在这方面表现良好，但在实际中不一定就有那么优越的表现。

三、推荐系统本身面临的问题

由于作者是从推荐系统作为切入点，进行的模型构建，但我认为一些推荐系统会遇到的问题并没有在文章中有所体现，比如我们在机器学习课程学习中，老师多次提及，推荐系统的 **user-item** 矩阵会面临

几大问题，其中一个是冷启动问题，一个是矩阵稀疏问题，还有就是可扩展性。由于缓存，尤其是应用到 CDNs 的缓存需要面临大量的网络流量，需要存储大量的流形等数据，数据规模一大，上述的这几个问题都容易导致构建的 DCL 表现不佳，比如对于一个新的用户，我应该如何知道他对于不同内容的打分、从而拟合他的喜好曲线呢？还有就是对于大量的用户，以本文使用的电影评分作为例子，电影是海量的，可是每个用户看的数目其实相对是少很多的，那么很多电影都没有被用户评分，那我们又该如何应对这种情况呢、主动地选择内容缓存呢？

从这个思考维度，我还发现，其实我们看电影，需求上是比较少的，并且电影缓存的特点是，每次缓存的部数比较少，一个用户可能一段时间内就缓存几部，这点十分不同于日常“网页”、“短视频”等这种短时间、大量的缓存情况；而且有的用户观影时间一般都是每周的周末或者某个时间段，那么我们为了这些特定用户而去提前“主动”缓存资源，我个人认为是有些不合理的，因为这会造成缓存的浪费。

还有一个值得注意的是，对于推荐系统，用户的喜好其实很难预测的，因为用户具有时间性，可能最近一段时间风靡这个，下一段时间就不热门这些内容，因此这对于“主动预测缓存”也是一个挑战。

但总的来说，我认为本文的创新点还是可圈可点，把推荐系统的思想用到了缓存上，引入了“主动预测”这个理念我觉得还是非常不错的创新，因为传统缓存的设计一般都是针对于那些已经被调用至内

存的数据的进出规则，而随着机器学习的进展以及一些预测任务的突破，我认为缓存的“预测”确实是一个值得研究的地方。

关于本文的改进，我认为可以继续在一些其他的推荐系统数据集中进行模型的测试，同时另一方面，在结合了神经网络的推荐系统相关任务中其实也有许多改进与优化的策略，我认为本文的模型还可以有进一步的模块搭建。比如说，我觉得可以不完全依赖于神经网络的预测，也就是说我们的缓存可以不仅仅只有“主动 **proactive**”的部分，还应该结合一些实际的情况来缓存——可以让缓存的一半空间，采用比较先进的传统缓存策略，然后另一半用来预测用户的喜好内容并主动地提前缓存。当然在这种想法中，需要进一步减小预测内容时候的模型规格，这样才不至于在大范围应用的时候模型过于庞大、复杂。