背景：

主动内容缓存已被认为是一种很有前途的解决方案，可以应对使用无线和移动设备进行内容访问的快速激增所带来的挑战，并防止内容提供商遭受重大收入损失。

人均设备和连接数量预计将从 2018 年的 2.4 个增长到 2023 年的 3.6 个[12]。由于移动网络流量的激增，移动网络和物联网系统面临着巨大的挑战，尤其是与用户服务质量/体验（QoS/QoE）有关的挑战

CDN 依赖于将缓存版本的内容存储在多个地理位置，以应对巨大的需求增长，提高网络性能和用户的 QoE。在其他原语（如网络缓存）中，存储成为网络底层的一个组成部分。

研究问题：

对于内容缓存，主要问题是在缓存容量有限的情况下，缓存或收回哪个对象以及何时缓存或收回。由于缓存单元的存储限制，必须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（即按需缓存）。然而，一些最先进的 CDN 采用了反应式缓存方案的变体，例如先进先出（FIFO）、最近最少使用（LRU）和最不频繁使用（LFU）。然而，它们并不能预测项目未来的受欢迎程度，这通常会降低缓存系统的性能。

然而，由于与要提供的可用数据相比，缓存的容量固有地有限，过去几年的广泛研究致力于提出有效的缓存方案，负责决定存储哪些项目以及何时从缓存中清除。然而，与理论最优值（OPT）相比，仍有更大的改进空间。

最近，许多研究已经开始提出基于人工智能的缓存方案，以超越经典方法，更接近 OPT。最近的缓存方案结合了一种预测内容流行趋势的“主动”行为。[10]提出了一种基于协作过滤的小小区网络缓存算法。该算法对用户和项目特征之间的交互进行建模，并利用这些信息进行内容流行度预测。最近，通过使用深度神经网络学习用户项目交互，协作过滤得到了显著改进。这种神经协作过滤（NCF）[11]方法被应用于推荐系统领域。

采用方法：

本文提出了一个用于主动内容缓存的端到端深度学习框架，该框架对用户和内容项之间的动态交互，特别是它们的特性进行建模。

所提出的模型通过深度神经网络模型在每个用户的不同内容项之间建立概率分布来执行缓存任务，并支持集中式和分布式缓存方案。此外，本文还解决了一个关键问题：在内容缓存中，我们是否需要一个明确的基于用户-项目对的推荐系统？即，在解决内容缓存问题的同时，我们是否需要开发一个推荐系统？为此，文章中引入了一个端到端的深度学习框架。

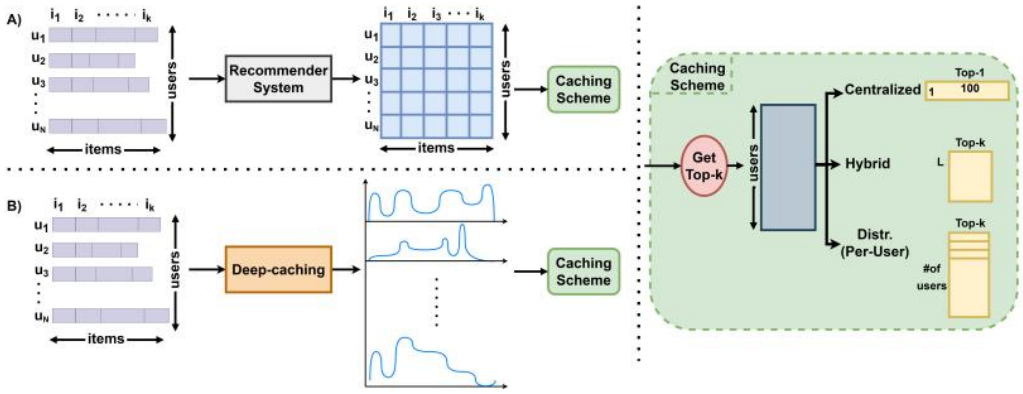
首先，在 NCF 模型的基础上，提供一个端到端的深度学习（DL）模型，以改进主动内容缓存。

所提出的模型通过对用户和项目特征之间的交互进行建模的深度神经网络，通过建立每个用户的内容项目的概率分布来执行缓存任务。与为解决内容推荐的二进制分类问题而训练的原始 NCF 模型不同，所提出的框架扩展了 NCF 模型以直接学习主动缓存决策。

（深度强化学习（DRL）是一种先进的基于 ML 的工具，已广泛用于网络内容缓存[19]。例如，在[6]中，为了激励深度学习用于边缘缓存，在移动边缘缓存的关键挑战和独特的 DRL 方面之间进行了系统映射。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；以及（ii）直接缓存学习者。

前一组估计特定缓存相关参数的值和演变，如内容流行度和缓存过期时间。同时，在后一类中，直接使用 DRL 来学习最佳缓存策略。属于第一类，在[3]中，提出了一个名为 DeepCache 的内容缓存框架。通过利用基于 LSTM 的模型来预测内容对象的流行程度，DeepCache 允许主动缓存未来最需要的项目。）

A. 两阶段缓存范式



如图 1-A 部分所示，内容缓存问题的一个可能公式包括将其分为两个主要模块：

1）推荐系统 f(M)，其中 f 是非线性函数，M 是用户-项目交互矩阵；

2）缓存方案：负责将矩阵 $M \in R_N \times K$ 编码为缓存矩阵 $H \in R_N \times K_h$，其中 N 是用户数量，K 是项目的数量，K_h是可配置的参数，表示每个用户基于他/她的偏好的首选项目的数量。

本质上，推荐系统以数据驱动的方式学习用户对特定项目的偏好。然后利用学习到的偏好来选择每个用户未来最喜欢的项目。给定包含 N 个用户和 K 个项目的数据，推荐系统的目标是生成矩阵 $M \in R_N \times K$。因此，推荐系统的主要目标是学习用户和项目之间、用户之间以及项目之间的相关性。

通常，我们将推荐系统公式化为，$M_0 = f(M)$，其中 f(M)是一个非线性函数，$M \in R_N \times K$ 是表示不同用户-项目对之间交叉交互的输入数据，M_0表示每个用户

-项目交互的预测得分。为了了解用户对特定项目的偏好，需要捕捉三个层次的相关性；1） 用户内相关性、2）项目内相关性和 3）用户项目互相关。

推荐系统训练的目标是学习这些相关性，以产生推荐矩阵 $M \in R_N \times K$，其中 m_{un}、i_k表示任意用户 un 和任意项目i_k之间发生交互的可能性的预测得分。根据上面的公式，推荐系统可以被解释为二元分类问题，其中对于每个用户项目对，推荐者旨在生成一个分数，该分数暗示该用户可能有多喜欢这个特定项目。

总述该算法流程：在训练阶段，给定用户-项目对，分类器预测这些交互可能发生的可能性。真实标签m_{un}、i_k是一个布尔指示器，其中 True 表示观察到用户和项目之间的交互，而 False 是由于可能导致其发生的广泛可能性而产生的误导信号（例如，用户不知道该项目的存在）。在推理阶段，这样的系统复杂度为 O(N * K)，其中 N 是用户数量，K 表示项目数量。为了解决内容缓存问题，在推荐模块之后堆叠了一个缓存块，将预测矩阵M_0编码为缓存矩阵 $H \in R_N \times K_h$。

B. 端到端缓存模式

与两阶段缓存模式(首先处理推荐任务，然后缓存基于推荐的内容)不同，我们提出了一个端到端框架，称为 DLC。在两阶段缓存范式中，网络架构和训练目标被设计用于处理不同的任务，即推荐系统，其中神经网络被训练来解决给定用户-项目交互的二进制分类问题。

受上述基本原理的启发，我们现在要解决的问题是:我们能否设计一个直接解决内容缓存问题的框架，而不是使用推荐系统作为中间阶段？ 为了回答这个问题，我们提出了一个端到端框架，通过利用神经网络架构来根据偏好学习每个用户的 top-k 项，直接处理内容缓存任务。给定交互矩阵 $m \in r_N \times K$ 中的单个用户项对，我们的目标是生成整个 K 项的概率分布。

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

制定训练目标以生成特定用户 un 与整个项目 K 之间的似然度量，使我们的体系结构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 un 与给定样本交互中未提及的其他项目之间的关系。

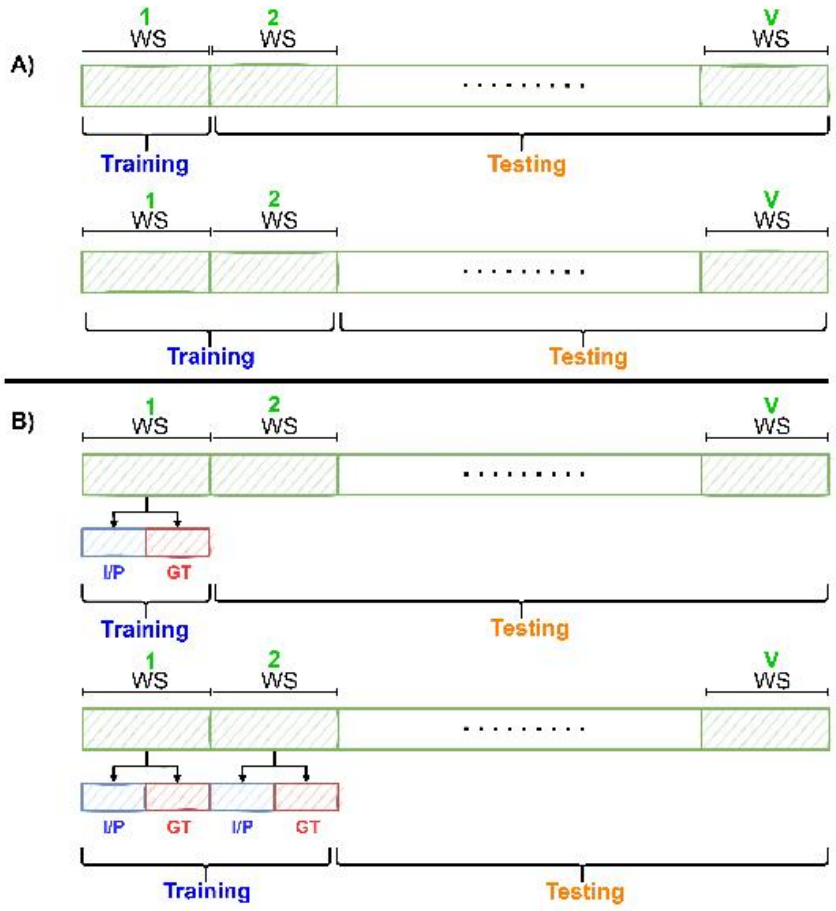
在推理过程中，我们将每个用户的特征提供给网络，反过来，网络生成该用户在数据中表示的整个项目中的偏好分布。

C.数据准备

我们的目标是开发一个现实的缓存框架，它不假设对未来数据的访问。为此，实现了一种窗口策略，将数据分成 V 个区间，如图所示。我们的缓存模块采用了一种现成的缓存技术，它不需要学习过程，只需要在第一个间隔中学习，同时存储在该间隔中发生的所有交互。

在端到端方法中，标签表示给定特定交互发生的情况下，特定用户与数据集中每个项目交互的概率。在这个关键差异的驱动下，在端到端情况下，需要在相同的间隔内进行另一次分割，以使用输入数据独立地准备标签，以避免网络可能

学习到的琐碎解决方案。换句话说，将输入交互与真值条件概率解耦对于防止网络在没有学习隐藏的用户项相关性的情况下记忆 top-k 项至关重要



训练 DLC 框架所需的数据准备概述。第 A 部分和第 B 部分分别演示了两阶段和端到端方法之后的窗口操作。将数据集分成相等的窗口，每个窗口有固定大小的 WS,V 表示间隔的个数。GT 表示 ground-truth 标签。

上半部分演示了两阶段方案中的窗口技术，其中数据被分割成 V 间隔以模拟现实场景。而下半部分则展示了端到端方案中的窗口技术，其中训练数据分为两部分;第一部分被馈送到网络，而第二部分被用来为每个用户生成整个数据集中所表示的整个项目的条件分布，而不是在此间隔内可访问的部分

D.缓存方案

无论采用哪种训练目标，无论是两阶段方案还是端到端方案，都需要一个缓存模块来生成 top-k 项K_h，在给定预测矩阵M_0的情况下，将其缓存到系统中，其中K_h是一个超参数，表示为每个用户选择的 top-k 项的数量。为此，如图 1 右侧所示，我们提出了三种方法将预测矩阵M_0映射到缓存矩阵 H。

集中式方法根据项目的频率对其进行排序，然后根据预测矩阵M_0获得所有用户的前 k 个项目。因此，缓存的矩阵 $H \in R^{1\times K_h}$在内存占用方面是最有效的。

每用户方法根据他们的预测分数对项目进行排序，并存储个性化的缓存版本，每个用户都有自己的缓存矩阵$H_u \in R^{1\times K_h}$。这种设置模仿了“分布式”设置，其中有一个处理数据的中心节点，但在最后，它为每个用户存储一个自定义的缓存矩阵。如果我们将每个用户的整个定制矩阵连接到中心节点，那么缓存的矩阵将是 $H \in R^{N\times K_h}$。

混合方法可能是效率和有效性之间的折中，其中缓存的矩阵 H 不会像每个用户方法那么大，也不会像集中式方法那么简单。在此设置中，我们将用户分成 L 组，其中对每个组应用集中式方案，从而得到$H_l \in R^{1\times K_h}$。然后通过将所有组连接在一起，最终缓存的矩阵将是 $H \in R^{L\times K_h}$。

环境配置：

在 windows 环境下打开 anaconda prompt，cd 到所要配置虚拟环境的文件夹，

依赖的下载：

* python==3.7

* pandas==0.24.2

* numpy==1.16.2

* pytorch==1.7.1

* gensim==3.7.1

* tensorboardX>=1.6 (mainly useful when you want to visulize the loss, see https://github.com/lanpa/tensorboard-pytorch)

使用 pip 下载这些依赖。

将 yml 文件放到前面创建的文件夹中，将文件中每一个库第二个等号及后面的配置信息删除，将文件 C:/user/your_username/.condarc 中的信息更改为清华源 ，更改如下：

channels:

- https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/free/

- https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/menpo/

- https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/bioconda/

- https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/msys2/

- https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/conda-forge/

- https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main/

- defaults

show_channel_urls: true

```
name: content_caching
channels:
  - pytorch
  - defaults
dependencies:
  - _libgcc_mutex=0.1=main
  - _openmp_mutex=4.5=1_gnu
  - backcall=0.2.0=pyhd3eb1b0_0
  - blas=1.0=mkl
  - ca-certificates=2021.7.5=h06a4308_1
```

更改为：

```
name: content_caching
channels:
  - pytorch
  - defaults
dependencies:
  - _libgcc_mutex=0.1
  - backcall=0.2.0
  - blas=1.0
  - ca-certificates=2021.7.5
  - certifi=2021.5.30
```

再使用 conda create env export content_caching.yml。若没有报错则成功安装，若出现依赖项报错，则再将 yml 文件中的依赖项先删去，成功下载后再使用 pip 或 conda install 安装。

注意使用校园网安装时可能出现以下报错：

```
(base) D:\python\pycharmproject\architecture\Proactive-Content-Caching-with-Deep-Learning-main>conda create -n content_c
aching_test -f content_caching1.yml
Collecting package metadata (current_repodata.json): failed

CondaHTTPError: HTTP 000 CONNECTION FAILED for url <http://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main/win-64/curren
t_repodata.json>
Elapsed: -

An HTTP error occurred when trying to retrieve this URL.
HTTP errors are often intermittent, and a simple retry will get you on your way.
'http://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main/win-64'
```

如果出现，则可以使用个人热点进行下载。

```
(base) D:\python\pycharmproject\architecture\Proactive-Content-Caching-with-Deep-Learning-main>conda env create -n conte
nt_caching_test -f content_caching1.yml
Collecting package metadata (repodata.json): done
Solving environment: failed

ResolvePackageNotFound:
  - libuuid=1.0.3
  - gstreamer=1.14.0
  - torchvision=0.2.2
  - libstdcxx-ng=9.3.0
  - libxcb=1.14
  - ld_impl_linux-64=2.35.1
  - ncurses=6.2
  - glib=2.69.0
  - dbus=1.13.18
  - libgomp=9.3.0
  - _openmp_mutex=4.5
  - gst-plugins-base=1.14.0
  - readline=8.1
  - libgcc-ng=9.3.0
```

当出现此时的错误时，将 yml 文件中对于的依赖项删去，等待顺利安装完成后再逐个进行 pip 安装。（或者再代码运行时，通过查看哪一个库无法 import 再逐个安装）。

如果这种方法仍在安装时产生过多库冲突，则直接创建 content_caching 环境，根据 main 运行时缺少的库与本地环境进行逐个安装。

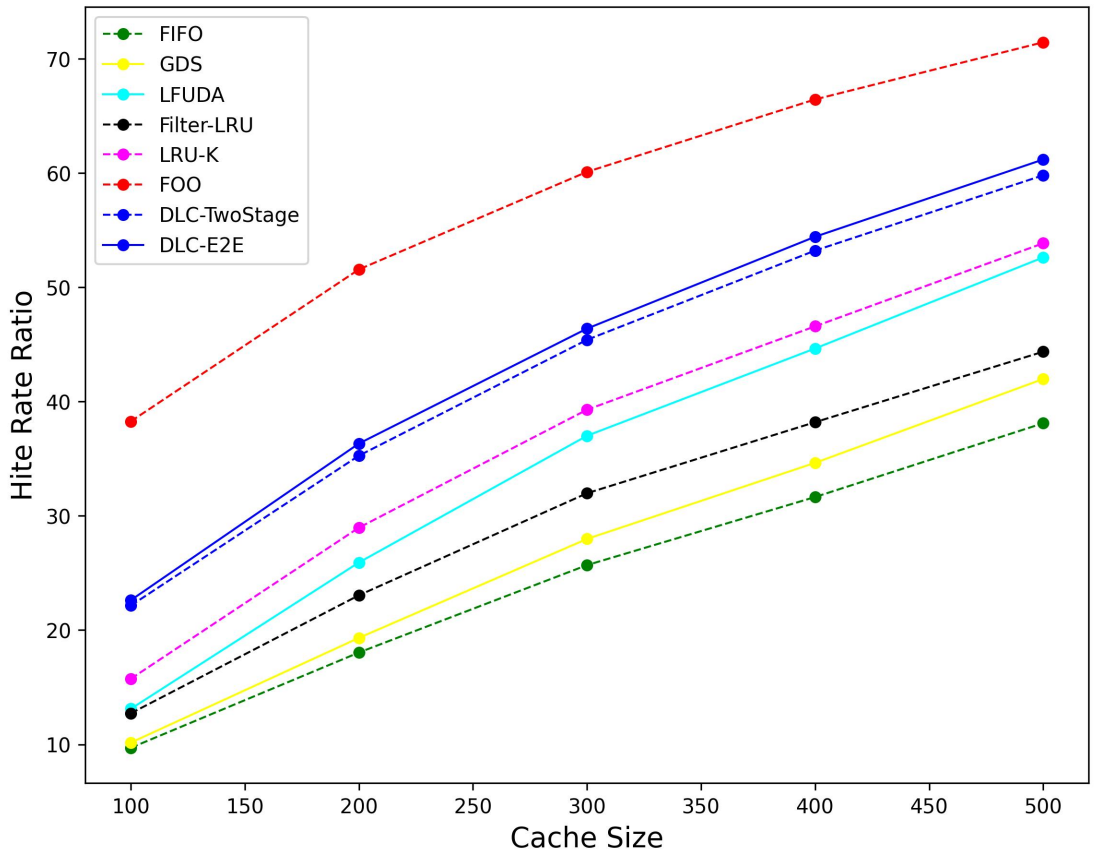
在这里下载训练所需的数据。

Download the data from this link and put the "data" folder under DLC folder.

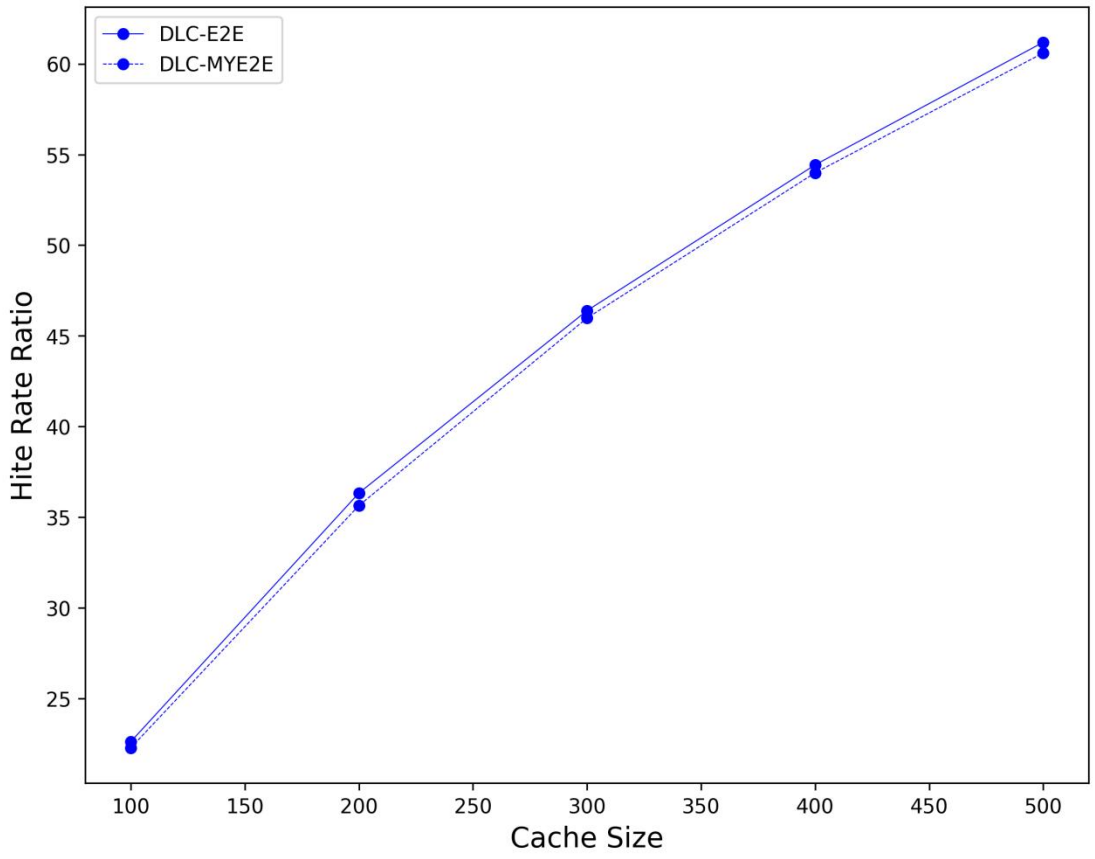
以下提供我本地 conda 环境的 yml 文件。mycontent_caching.yml

实验结果：

论文中将 E2E 算法和其它常见的 Cache 算法进行对比，生成如下图像：



这里实现了 E2E 算法在 ml-1m 数据集上的复现。在其他缓存算法上的复现由于数据集无法迁移而未能实现。因此仅生成 E2E 算法对比的图像，如下所示：



生成数据的结果：best epoch 005: HR = 0.603, NDCG = 0.358，在本地复现的结果与论文相近。
具体生成的数据结果存放在 data.txt 文件中。