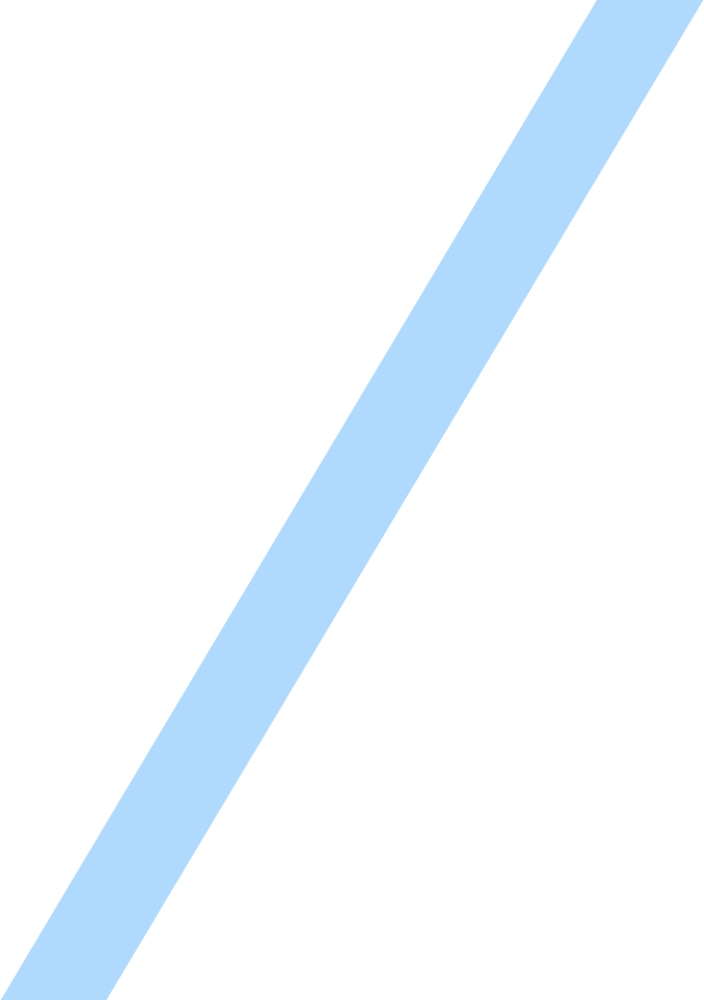
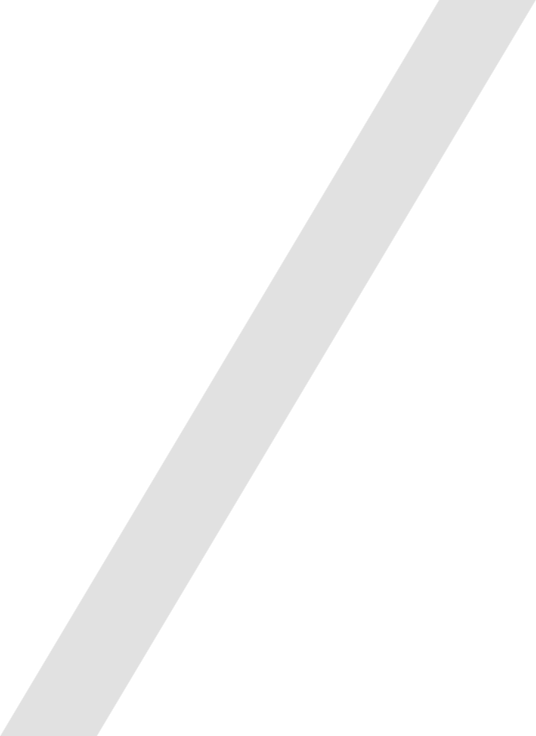
|  |
| --- |
| Deep Learning Recommendation Model - DLRM |

|  |
| --- |
|  |



|  |  |
| --- | --- |
|  |  |



Saioa Prieto Portugal, 633301

Alexander Pérez Bunea, 636393

Fabio Misino, 512377

Proyect for *SISTEMI INTELLIGENTI PER INTERNET* and *DEEP LEARNING*

Link to GitHub repository: [CODE](https://github.com/spriep/model_explanations_using_influence_analysis.git)

Content

Model’s architecture2

SETUP 3

Dependencies3

Docker3

TESTING WITH THE MODEL4

Dataset 4

Data Preprocessing5

Training6

Benchmarkig6

Testing6

CONCLUSION8

|  |
| --- |
| MODEL’S ARCHITECTURE Model’s summary SETUPDEPENDENCIES Summary of dependencies DOCKER ERROR → The torchrec-nightly library depends on fbgemm\_gpu, a native library that is only available on Linux, not on Windows. This library contains C++ and CUDA code that does not compile or run on Windows.  SOLUTION → Use a Docker container with a personalized image. This way: You only configure it once. You get a complete, reproducible, and portable environment. You avoid having to manually install packages every time you run the container.  I created the custom image, first creating a docker file with all the dependencies and the running the following command:  **docker build -t torchrec\_custom**  Then to use it I just runed the command to activate it adding my path of the project so it can be used once inside the docker:  **docker run --gpus all -it --rm -v "your\_path:/workspace" torchrec\_custom bash**  IMPORTANT 🡪 three libraries usage come up later on the usage of the model so they have to be installed manually once the docker is activated: **mlperf-logging, tensorboarda nd numactl** TESTING WITH THE MODEL If you run the commands by themselves they are set up to run with random automatically generated data and with heavy settings done to be run on terminals with several gpus witch ends up on the processes being killed as the RAM and GPU memory runs out. |

### DATASET

[Kaggle Display Advertising dataset](https://go.criteo.net/criteo-research-kaggle-display-advertising-challenge-dataset.tar.gz)

### DATA PREPROCESSING

I took the train.txt from the zip.

Data must undergo a preprocessing to being able to enter the model.

The procecess splits the data on “days” and process each day separatlly, the program reads all the dataset and then divides the data equally among the days, and each day has a minimum amount of data approximately 1,28 millions of entries, it loads each split, reads all its entries saving also which entries are labelled as 1 and 0, and then stored in files as train\_day\_X.npz, at the end of the process the program take all this generated sub-files and converge all of them in a general file named kaggleAdDisplayChallenge\_processed.npz. I did the preprocessing of the 45 million data entries (I think it was this value I didn’t check), wich took 12 hours for the process to be killed when the program wanted to concatenate all the splits for making a final all-together file.

So after many tries I discovered that the training has a minimum amount of “days” to be able to conclude with out errors, which is 7 so, I sampled 9 million entries from train.txt (using a head from the train.txt and redirecting it to another txt file), which is roughly the minimum required to cover 7 days.

To run the preprocess you must enter in the bash the following code:

python

import os

from data\_utils import getCriteoAdData

getCriteoAdData(

    datafile="/workspace/dlrm/datasets/train\_9M.txt",

    o\_filename="kaggleAdDisplayChallenge\_processed",

    max\_ind\_range=9000000,

    sub\_sample\_rate=0.0,

    days=7,

    data\_split='train',

    randomize='total',

    criteo\_kaggle=True,

    memory\_map=False

)

### TRAINIG

As said before the model was extremely heavy, I did so many tries all ending on the killing of the process. On the command used to execute the training all the “light” parameters can be seen:

python dlrm\_s\_pytorch.py \

  --arch-sparse-feature-size=8 \

  --arch-mlp-bot="13-8-8" \

  --arch-mlp-top="8-4-1" \

  --data-generation=dataset \

  --data-set=kaggle \

  --raw-data-file=./datasets/train\_9M.txt \

  --processed-data-file=./datasets/kaggleAdDisplayChallenge\_processed.npz \

  --loss-function=bce \

  --round-targets=True \

  --learning-rate=0.05 \

  --mini-batch-size=1 \

  --test-mini-batch-size=2 \

  --test-num-workers=0 \

  --num-batches=10 \

  --print-freq=10 \

  --print-time \

  --test-freq=10 \

  --save-model=dlrm\_model.pth

Important note: We perform an accuracy check (if it's better than the previous training, it gets saved) so the trained model can be stored in first place and after if a training with better results occurs it also saves it.

TRAINIG OUTPUT:

*world size: 1, current rank: 0, local rank: 0*

*Using CPU...*

*Reading pre-processed data=./datasets/kaggleAdDisplayChallenge\_processed.npz*

*Sparse fea = 26, Dense fea = 13*

*Defined train indices...*

*Randomized indices across days ...*

*Split data according to indices...*

*Reading pre-processed data=./datasets/kaggleAdDisplayChallenge\_processed.npz*

*Sparse fea = 26, Dense fea = 13*

*Defined test indices...*

*Randomized indices across days ...*

*Split data according to indices...*

*time/loss/accuracy (if enabled):*

*Finished training it 10/10 of epoch 0, 228.55 ms/it, loss 0.675228*

*Testing at - 10/10 of epoch 0,*

*accuracy 90.000 %, best 90.000 %*

*Saving model to dlrm\_model.pth*

If we want to make predictions using the trained model we saved, add these parameters:  
--inference-only --load-model=dlrm\_model.pth

### BENCHMARKING

For performing the benchmarking we use the code specifically for this datset.

The benchmarking is runed both on CPU and GPU generating each own output files for checking the results.

The output files consists on:

(.prof/.json) 🡪 These are files generated by the profiler (a tool that measures code performance, timing, and resource usage). They contain technical data for detailed performance analysis.

* Two .prof files 🡪 It includes detailed performance metrics for both CPU and CUDA (GPU) operations.
* One .json file 🡪 It contains both technical information about the GPU where the code was executed and a list of execution events, each representing an operation on the CPU or GPU during training or inference.
  + It can be visualized in: [chrome://tracing](chrome://tracing/)
  + A screenshot of a computer

    AI-generated content may be incorrect.
  + A screenshot of a computer screen

    AI-generated content may be incorrect.
* One .log file 🡪 contains the output of the program, in this case a training

Important note: at the moment of running the benchmark error shows on the interface saying the .prof and .json files cannot be found as they are not created in the same folder as the program, but they are safe and well created.

Command: ./bench/dlrm\_s\_benchmark.sh "--enable-profiling --save-model=dlrm\_model.pth"

### TESTING

In the official testing file the following error arose*: bash: ./dlrm\_s\_test.sh: /bin/bash^M: bad interpreter: No such file or directory*. Wich means that the *dlrm\_s\_test.sh* file was created or edited on Windows, and contains Windows-style line endings (^M), which breaks the script in Unix/Linux environments.

Solution 🡪 Convert the file to Unix format.

With the command: dos2unix dlrm\_s\_test.sh

(dos2unix needs to be instaled in the docker)

NEXT STEPS:

* I want to run inference (testing to see how it works):  
  I take the last 1M entries from the TRAIN.TXT file to use as TEST.TXT.  
  I preprocess TRAIN.TXT, load the data and the pre-trained model into dlrm\_s\_pytorch.py with inference, and run the inference.
* RUN THE TESTS GIVEN ON THE OFFICIAL REPOSITOTY

CONCLUSION