Project 2: Containerization

by Nils Schell (nils.schell@stud.hslu.ch)

This report outlines the containerization of the Project 1 distilbert-base-uncased model for the MRPC paraphrase detection task. Through a systematic process using Docker, GitHub Codespaces and Weights & Biases (Wandb), a reproducible training environment was created. The main finding is that containerization ensures identical execution across different CPU environments (local vs. cloud), with only minor difficulties in ensuring smooth training runs on GPU and CPU hardware with the same code.

The task of the project required to convert the existing notebook into a runnable Python script. To enable persistent logging in a containerized environment, MLflow was replaced with Weights & Biases (Wandb). Wandb is designed for distributed runs and only requires an API key, passed as an environment variable. This makes it easier to use than the locally-hosted MLflow server for this use case, especially in the GitHub Codespace. At the beginning of the script, the Wandb API key environment variable is loaded from a .env file. To ensure local reproducibility, it is therefore necessary to create a .env file in the root folder and to include your Wandb API key in this file.

To speed up local training, the GPU torch version was chosen. To integrate GPU training, the correct torch CUDA hardware version must be installed on the environment, which can be found here: https://pytorch.org/get-started/locally/ . To find the correct CUDA version for your environment, simply type the command "nvcc --version" in a terminal. Due to the GPU support, a training run takes about 4 minutes. To start the training run simply type "python training.py --checkpoint_dir models --warmup_ratio 0.1 ...". If the arguments are not used, the default hyperparameter settings will be selected instead.

The primary goal of Task 2 was to create a minimal, portable Docker image that encapsulates the training run. Based on the project hints ("Docker Playground has very little memory"), the key decision was to build a CPU-only image to ensure portability and minimal size, rather than a large hardware-dependent GPU image. To illustrate the size of scale, the GPU image was 20.85 GB in size, whereas the CPU image was only 3.15 GB. This is an 85% decrease in size. The training speed on the CPU based Docker environment was 17 minutes this time, which is four times slower than the GPU version.

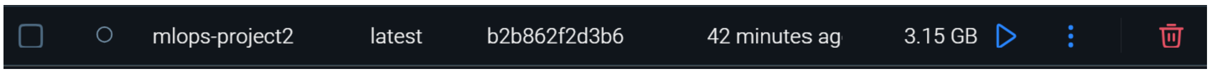

mlops-project2 latest b2b862f2d3b6 42 minutes ag 3.15 GB

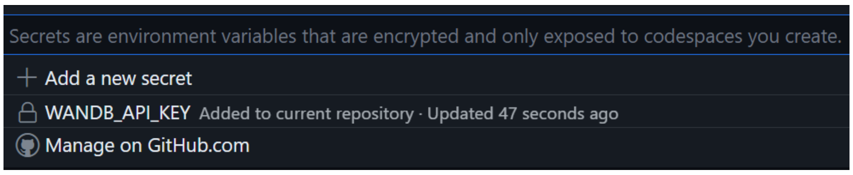
Figure 1: CPU-based Docker image

So it's a trade-off between size and speed in this use case. To guarantee smooth operation during local GPU training and afterwards CPU containerisation, two things are required:

1. With the training option accelerator="auto", the script will select the optimal hardware for the environment before training begins. If a GPU is available, it will use the GPU for training. If only a CPU is available in the environment, the CPU will be selected. This ensures smooth training on whatever hardware is available.
2. When building the Docker image, make sure that the right torch hardware version is used. On a system equipped with a GPU, it sometimes automatically selects the GPU torch version when building the docker image. In this case it's an unusable Docker image for cloud services with low memory as Docker Playground. Therefore, two small adjustments were needed. Remove 'torch' and 'lightning' from the requirements.txt file and install the CPU version in advance. The following two lines were added to the Dockerfile to force this correct installation:
   ```
   RUN pip install torch --index-url https://download.pytorch.org/whl/cpu
   RUN pip install lightning
   ```

To automate the build and execution, a helper script run_docker.py was created. This script automatically passes the WANDB_API_KEY from the environment into the container, executing the full training run with a single command. This was tested using the optimal hyperparameters from project 1 and achieved the same F1 score.

To verify the portability of the image, it was also run on GitHub Codespaces. To secure the Wandb API Key and still make a smooth pipeline run with run_docker.py, the key was securely supplied using GitHub Codespace Secrets, which automatically populates it as an environment variable in the terminal. The Wandb API key was not stored in the repository because the .env file, which includes the key, was added to the .gitignore file. The same hyperparameters were used as before with the local Docker image, so the outcome was the same. This proves the portability of the Docker images. Due to the limited 2-core CPU processing power available on Codespace, the entire training run took about 47 minutes. Compared to the 4 minute GPU execution, this is almost 12 times slower.

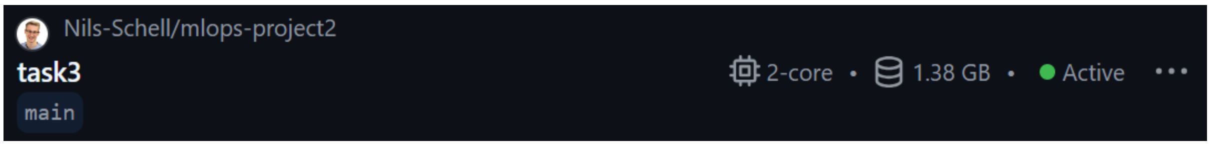

Secrets are environment variables that are encrypted and only exposed to codespaces you create.

Add a new secret

WANDB_API_KEY Added to current repository · Updated 47 seconds ago

Manage on GitHub.com

Figure 2: Wandb API key added to the GitHub Codespace Secrets


Nils-Schell/mlops-project2

task3

main

2-core • 1.38 GB • Active

Figure 3: Codespace processing power

Finally, all the project files were organised in a public GitHub repository. These included the Dockerfile, training.py, requirements.txt and the run_docker.py helper script. As required by Task 4, a README.md file was added to describe the project structure and provide clear, step-by-step instructions for smoothly cloning, building and running the containerised application. Link to the public GitHub repository: https://github.com/Nils-Schell/mlops-project2

Conclusion:

The most interesting finding was the reproducibility of the CPU-based container. Identical F1 scores of ~0.9019 were achieved on both a local Windows machine and a remote Linux Codespace, proving that the container successfully homogenized the development environment, which was the project's primary goal. The slight performance drop in the GPU run (~0.8881) is because the optimal hyperparameters from the initial project were not selected for this test run. I will select them for a better comparison next time. In retrospect, the Docker build process was more complex than anticipated because of the GPU/CPU environment decision. I wanted to make sure that the system could automatically select the optimal hardware to guarantee a certain degree of hardware flexibility. The multi-step "pip install" strategy was a necessary workaround to force a CPU-only torch installation. A multi-stage Docker build could potentially have streamlined this process, but the three-step RUN process proved effective. Future iterations would benefit from exploring multi-stage builds to further optimize image size and the build process for each system.

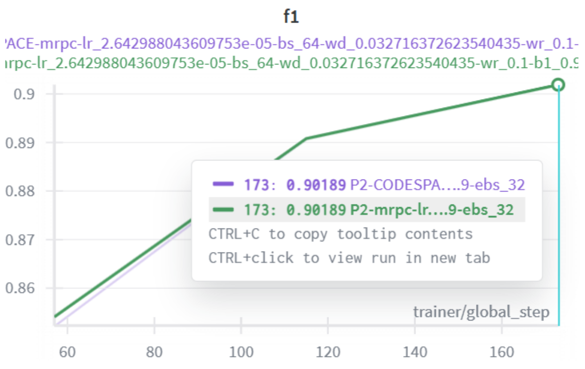
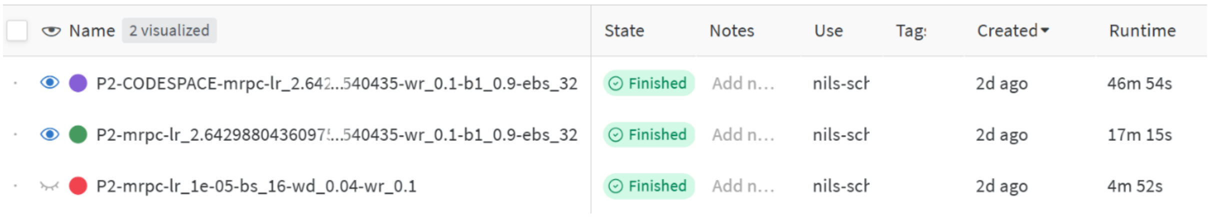


Figure 4: Identical F1 score



Name 2 visualized	State	Notes	Use	Tag	Created	Runtime
P2-CODESPACE-mrpc-lr_2.642…540435-wr_0.1-b1_0.9-ebs_32	Finished	Add n…	nils-sch		2d ago	46m 54s
P2-mrpc-lr_2.6429880436097…540435-wr_0.1-b1_0.9-ebs_32	Finished	Add n…	nils-sch		2d ago	17m 15s
P2-mrpc-lr_1e-05-bs_16-wd_0.04-wr_0.1	Finished	Add n…	nils-sch		2d ago	4m 52s

Figure 5: Wandb Logging of three Tasks with Runtime