Generative AI Usage Journal - ARI3129

Daniel Simon Galea, Liam Azzopardi, Liam Jake Vella, Matthew Privitera

January 31, 2026

Contents

1 Introduction

This project made use of several generative AI models, namely ChatGPT 5.2, Google Gemini 3, Claude Sonnet 4.5 and NotebookLM. These models were selected due to their reliability, robustness, and versatility across a wide range of tasks relevant to the project. Claude Sonnet 4.5 was used during the programming and development stages as it is particularly-suited for coding-related tasks such as debugging, code optimisation and explaining and outputting complex programming concepts. The ability to generate context-aware code makes it a valuable tool especially in the technical side of this project.

Google Gemini and ChatGPT 5.2 were used more broadly throughout the project. Google Gemini 3 was used due to the fact that the pro version is offered for free to students by the University of Malta directly making it a readily available powerful tool. ChatGPT proved useful due to its flexibility and ability to be used for many different tasks.

The use of generative AI models allowed for an approach where each model could be used in a way that would exploit it's strength and would overall increases the efficiency and quality of the project.

2 Ethical Considerations

Although generative AI tools were used to assist us in this project, their use raises important ethical considerations that must be addressed regarding data bias, privacy, academic integrity and transparency.

2.1 Data Bias and Fairness

When using GenAI tools to help us analyse experimental results, there is a risk that the AI's interpretations may reflect biases present in its training data. For example, when asked to suggest which important plots should be included in the evaluation section, the AI initially recommended a comprehensive six-plot layout typical for research papers. However, this recommendation may have reflected a bias toward more complex projects that would have been inappropriate for the scope of this one.

2.2 Privacy and Data Protection

The use of GenAI tools requires careful consideration of what information is shared. While data such as training logs and evaluation data is not personally sensitive, it is important to remember that GenAI tools retain conversation history, meaning project details may persist in external systems. To address these concerns, no personally identifiable information beyond standard result metrics were shared. The traffic sign images themselves were never uploaded in raw form. Furthermore, when seeking help with code, complete datasets were not provided, but instead only a small sample was given so that we can have a better understanding of how to evaluate the results.

2.3 Academic Integrity

The most significant ethical consideration involves maintaining academic integrity when AI tools are able to assist with technical writing and analysis. In this project, GenAI tools were used as an assistive tool rather than a content generator. Specifically, they were used to help improve text clarity, debug code errors, explain technical concepts and suggest plot layouts. However, all AI generated text was critically reviewed, edited, and adapted to reflect independent understanding of the project. Apart from that, AI provided explanations of concepts or models were cross-referenced with official documentation and research papers. AI suggestions were treated as a starting point for exploration rather than final answers. This was done to make sure that the final work reflects genuine understanding rather than unverified AI output.

2.4 Transparency and Accountability

Transparency and accountability of AI usage will be highlighted across the next sections, where we will describe the methodology, prompts and responses, improvements, errors and contributions made, and an individual reflection for each group member.

3 Methodology

Throughout the assignment, different generative AI models were used as collaborative tools to provide assistance throughout the different stages of the project, including code debugging, understanding concepts and model architectures and implementation guidance.

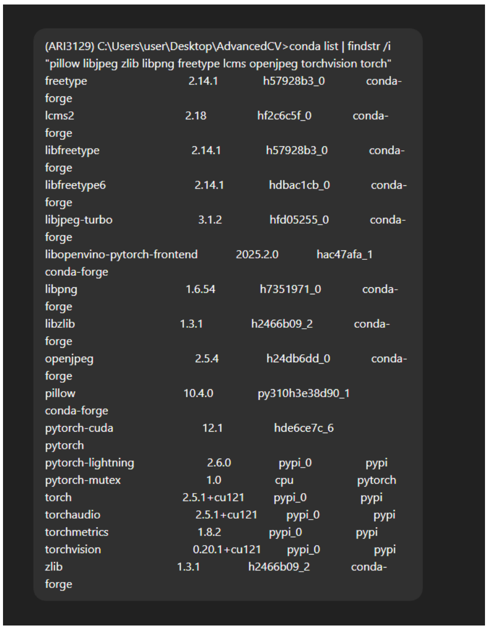
3.1 Environment Setup

Claude Sonnet 4.5 and ChatGPT-5.2 were particularly used to aid in the environment setup and help manage any version dependencies for the different python libraries that were needed to train the different models. Since there were some issues with the initial environment setup, Claude Sonnet 4.5 provided an identification of any required libraries and their appropriate versions to replace any library installation errors in the original *Dependencies.yml*. ChatGPT-5.2 helped in troubleshooting installation errors related to CUDA and Claude Sonnet 4.5 provided a new *Dependencies.yml* that successfully helped us setup the conda environment.

3.2 Implementation Guidance

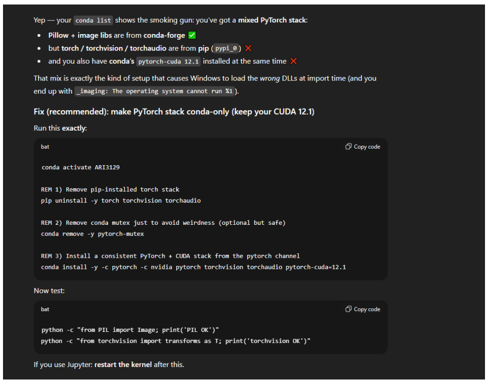
For the implementation process, Claude Sonnet 4.5 was selected based on its specific strengths to optimise different aspects of the code and to understand the effect different hyperparameter configurations have on the overall performance of the different models. Claude Sonnet 4.5 suggested the removal and addition of some hyperparameters, suggesting the inclusion of *warmup_epoch* and using *optimizer=auto* for YOLOv8, for example. As part of the implementation, Claude Sonnet 4.5 and NotebookLM helped us understanding the architectural concepts of each model when needed for example Focal Loss in RetinaNet and the ViT backbone architecture in RF-DETR. Specifically, NotebookLM was used by uploading research papers of different models to summarise different concepts and different architectures used in the assignment.

4 Prompts and Responses



```
(ARI3129) C:\Users\user\Desktop\AdvancedCV>conda list | findstr /i
"pillow libjpeg zlib libpng freetype lcms openjpeg torchvision torch"
freetype                  2.14.1            h57928b3_0        conda-
forge
lcms2                     2.18              hf2c6c5f_0        conda-
forge
libfreetype               2.14.1            h57928b3_0        conda-
forge
libfreetype6              2.14.1            hdbac1cb_0        conda-
forge
libjpeg-turbo             3.1.2             hfd05255_0        conda-
forge
libopenvino-pytorch-frontend   2025.2.0     hac47afa_1
conda-forge
libpng                    1.6.54            h7351971_0        conda-
forge
libzlib                   1.3.1             h2466b09_2        conda-
forge
openjpeg                  2.5.4             h24db6dd_0        conda-
forge
pillow                    10.4.0            py310h3e38d90_1
conda-forge
pytorch-cuda              12.1              hde6ce7c_6
pytorch
pytorch-lightning         2.6.0             pypi_0            pypi
pytorch-mutex             1.0               cpu               pytorch
torch                     2.5.1+cu121       pypi_0            pypi
torchaudio                2.5.1+cu121       pypi_0            pypi
torchmetrics              1.8.2             pypi_0            pypi
torchvision               0.20.1+cu121      pypi_0            pypi
zlib                      1.3.1             h2466b09_2        conda-
forge
```

Figure 1: Example of a Generative AI Prompt



Yep — your `conda list` shows the smoking gun: you've got a **mixed PyTorch stack**:

- **Pillow + image libs** are from **conda-forge** ✅
- but **torch / torchvision / torchaudio** are from **pip** (`pypi_0`) ❌
- and you also have **conda's** `pytorch-cuda 12.1` installed at the same time ❌

That mix is exactly the kind of setup that causes Windows to load the *wrong* DLLs at import time (and you end up with `_imaging: The operating system cannot run %1`).

Fix (recommended): make PyTorch stack conda-only (keep your CUDA 12.1)

Run this **exactly**:

```bat
conda activate ARI3129

REM 1) Remove pip-installed torch stack
pip uninstall -y torch torchvision torchaudio

REM 2) Remove conda mutex just to avoid weirdness (optional but safe)
conda remove -y pytorch-mutex

REM 3) Install a consistent PyTorch + CUDA stack from the pytorch channel
conda install -y -c pytorch -c nvidia pytorch torchvision torchaudio pytorch-cuda=12.1
```

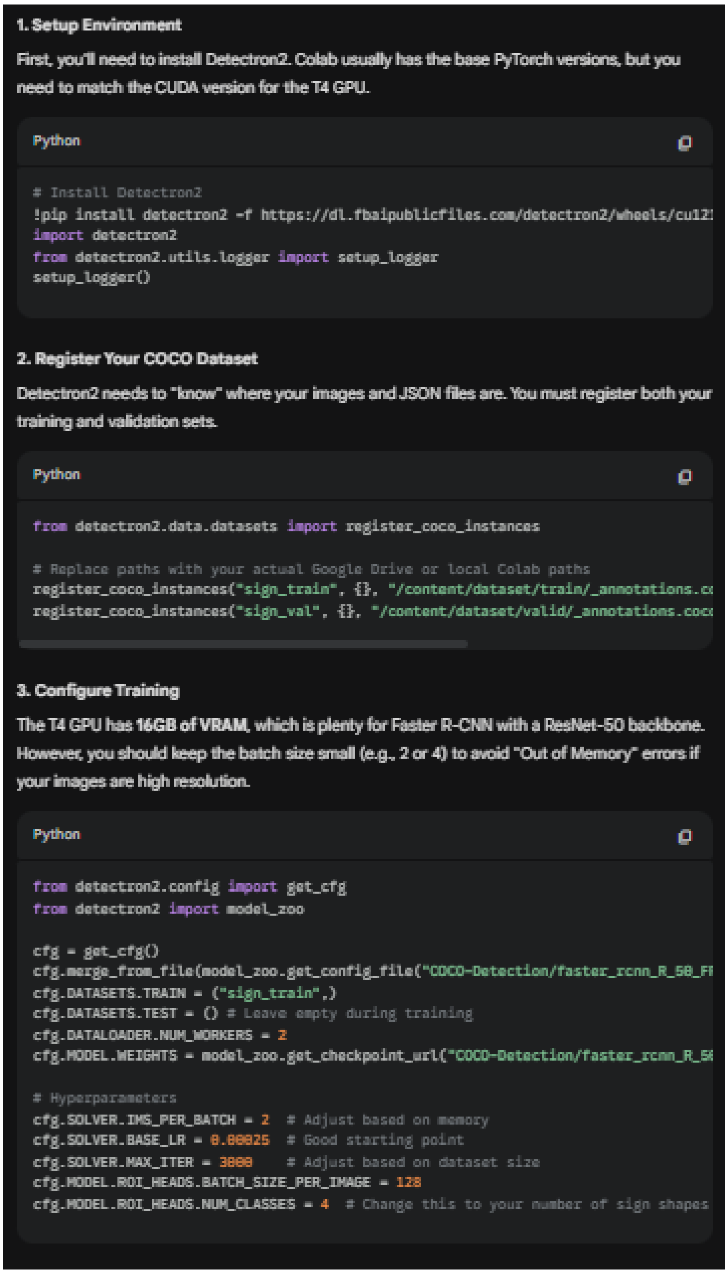
Now test:

```bat
python -c "from PIL import Image; print('PIL OK')"
python -c "from torchvision import transforms as T; print('torchvision OK')"
```

If you use Jupyter: **restart the kernel** after this.

Figure 2: Generative AI Response

In this prompt, ChatGPT helped me fix and restore my environment as my device had a windows update which in turn broke certain libraries in the environment. By giving the entire list of libraries in the environment, ChatGPT was able to pinpoint the problem and resolve the issues that I had in the environment so that I could resume my work by removing the libraries which were now incompatible with the new update and install the working libraries in their place that worked.



1. Setup Environment

First, you'll need to install Detectron2. Colab usually has the base PyTorch versions, but you need to match the CUDA version for the T4 GPU.

Python

```python
# Install Detectron2
!pip install detectron2 -f https://dl.fbaipublicfiles.com/detectron2/wheels/cu12
import detectron2
from detectron2.utils.logger import setup_logger
setup_logger()
```

2. Register Your COCO Dataset

Detectron2 needs to "know" where your images and JSON files are. You must register both your training and validation sets.

Python

```python
from detectron2.data.datasets import register_coco_instances

# Replace paths with your actual Google Drive or local Colab paths
register_coco_instances("sign_train", {}, "/content/dataset/train/_annotations.co
register_coco_instances("sign_val", {}, "/content/dataset/valid/_annotations.coco
```

3. Configure Training

The T4 GPU has **16GB of VRAM**, which is plenty for Faster R-CNN with a ResNet-50 backbone. However, you should keep the batch size small (e.g., 2 or 4) to avoid "Out of Memory" errors if your images are high resolution.

Python

```python
from detectron2.config import get_cfg
from detectron2 import model_zoo

cfg = get_cfg()
cfg.merge_from_file(model_zoo.get_config_file("COCO-Detection/faster_rcnn_R_50_F
cfg.DATASETS.TRAIN = ("sign_train",)
cfg.DATASETS.TEST = () # Leave empty during training
cfg.DATALOADER.NUM_WORKERS = 2
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-Detection/faster_rcnn_R_5

# Hyperparameters
cfg.SOLVER.IMS_PER_BATCH = 2  # Adjust based on memory
cfg.SOLVER.BASE_LR = 0.00025  # Good starting point
cfg.SOLVER.MAX_ITER = 3000     # Adjust based on dataset size
cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 128
cfg.MODEL.ROI_HEADS.NUM_CLASSES = 4  # Change this to your number of sign shapes
```

Figure 3: Example of a Generative AI Prompt

5 Improvements, Errors, and Contributions

Generative AI tools significantly enhanced various aspects of the project, from idea generation to debugging, while also presenting challenges that required verification and correction. These tools contributed to overall efficiency but required careful management to mitigate inaccuracies.

In terms of improvements to quality and efficiency, generative AI excelled in code-related tasks. For example, during model implementation, Claude was used to debug issues in object detection architectures, such as resolving incorrect configurations in training pipelines. By providing error logs and code snippets as prompts, the AI suggested optimisations, which reduced training time and improved model stability. Similarly, AI-assisted code generation for techniques like image rotations and mosaics, tailored to the dataset's real-world variations, leading to better generalisation across attributes like viewing angles or sign conditions.

Generative AI also helped in idea generation and literature review. For the background on object detection techniques, prompts were used to summarise architectural evolutions in models like the YOLO family, yielding overviews with key references that were adapted into the report. This streamlined research, saving time on sourcing and understanding information from recent publications.

However, errors and misleading information were introduced in some outputs. In one literature review instance, ChatGPT incorrectly described YOLOv11 as a community driven update. This mistake was identified by cross-checking with official documentation and corrected to ensure accuracy. Another error occurred in debugging suggestions, where incompatible configurations were recommended, leading to temporary performance drops in metrics. Validation runs and dataset-specific adjustments resolved this. In privacy discussions, AI sometimes exaggerated risks, requiring additional reviews that ultimately reinforced data protection practices but caused minor delays.

Overall, AI's contributions in areas like debugging and the literature review, boosted project quality, with errors addressed via human oversight, primary source verification and testing. This approach maximised benefits while maintaining reliability.

6 Individual Reflection

6.1 Daniel Simon Galea

From using these generative AI tools, I gained a deeper understanding of how different neural network architectures operate and how their respective hyperparameters influence model performance. Generative AI was particularly valuable in explaining neural network architectures, allowing me to grasp underlying concepts more effectively.

Generative AI significantly improved my overall efficiency, as it could be applied across multiple stages of the project. Its ability to debug code, suggest improvements and clearly explain concepts related to the various architectures I implemented reduced development time and minimised trial-and-error. This allowed me to focus more on refining my models and improve performance than on resolving avoidable technical issues.

The task that benefited the most from AI assistance was hyperparameter tuning. Generative AI provided guidance on selecting appropriate parameter ranges and optimisation strategies, enabling us to fine-tune my models more effectively. As a result, model performance and accuracy were improved, contributing to stronger results and a higher-quality final project overall.

6.2 Liam Jake Vella

My perspective on using Generative AI in academic projects has evolved from viewing it as a simple shortcut to seeing it as a sophisticated collaborative tool. Gen AI helped streamline the setup of the Detectron2 framework for Faster R-CNN (Figure 3). Configuring the `register_coco_instances` and custom `cfg` parameters for the 5 specific classes was a part of the project where Gen AI was most helpful for rapid debugging and code refinement. I found that Gen AI saved me a significant amount of time rather than wasting it. Instead of spending hours troubleshooting, I could focus more on analysing results and fine-tuning hyperparameters.

In writing the report, Generative AI also helped me refine my language, making it easier to understand and straight to the point. As a group, we decided to use LaTeXto write it. As I have minimal experience in using LaTeX, I used Generative AI to help me format and use the appropriate functions. The knowledge that I gained from this will be very useful for future projects, such as my Final Year Project.

6.3 Matthew Privitera

Throughout this project, I primarily used Claude for technical explanations, data analysis and writing assistance. I also used NotebookLM to help me understand YOLOv12 and

RF-DETR architectures and how they improved on their predecessors. This experience provided valuable insights into both the benefits and limitations of AI tools in academic work.

GenAI tools improved efficiency in specific areas. Technical documentation that would normally take hours of reading was summarised into clear explanations in minutes. For example, understanding the RF-DETR architecture which involves concepts like DINOv2 backbones (LW-DETR improvement). This guided reading approach saved substantial time while ensuring I understood the core concepts. Code generation was another area where AI proved highly efficient. Tasks like parsing JSON files, creating LaTeX tables and generating visualisations were completed in minutes. The AI could produce working Python scripts that I could test and modify, which eliminated the tedious process of going through long documentation.

However, AI usage also created inefficiencies in some cases. The AI often provided overly comprehensive responses when concise answers would suffice. For instance when asking about what type of metrics/plots would be appropriate for this assignment, it suggested a six-plot layout with detailed implementation code. This required time to read through and evaluate, only to decide that a simpler two-figure approach was more appropriate and effective. Additionally, while Claude is a very powerful tool, clarifying questions sometimes required multiple rounds of back-and-forth because it would overcomplicate the topic.

Overall, the integration of GenAI tools like Claude and NotebookLM proved to be an important asset in managing the technical complexity of this project. While the tendency for these models to provide overly verbose outputs required a critical and selective approach, their ability to simplify complex architectures like RF-DETR and automate boilerplate coding tasks was invaluable. These tools served as a bridge between high-level theoretical concepts and practical implementation to accelerate the development cycle while allowing more time for developing the coding architecture and thoughtful analysis.

6.4 Liam Azzopardi

Throughout the assignment, I mainly used Claude Sonnet 4.5 to understand technical concepts related with YOLOv8 and RetinaNet, and get aid in setting up the conda environment setup, debugging code and getting guidance on the implementation process. Using Claude Sonnet 4.5 throughout the assignment led to getting a better technical understanding about each model and the general implementation pipeline needed to implement such models, but with some notable limitations.

When asking for particular model information, Claude provided some misinformation within its response, making it important to cross reference with the official Ultralytics for YOLOv8 [1] and the research paper for RetinaNet by Lin et al. [2]. Misinformation included incorrect release dates and general assumptions including architectural details.

The use of generative AI helped in the efficiency of the project, where writing about technical concepts and individual model details proved to be much more efficient. Generative AI also proved to be useful in efficient code debugging with loading the training set, test set, and validation set into each model and general hyper-parameter suggestions to optimise accuracy but also manage limited computational resources.

7 References and Resources Used

References

[1] Ultralytics, *Explore Ultralytics YOLOv8*, en. Accessed: Jan. 31, 2026. [Online]. Available: `https://docs.ultralytics.com/models/yolov8/`.

[2] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, *Focal Loss for Dense Object Detection*, arXiv:1708.02002 [cs], Feb. 2018. DOI: `10.48550/arXiv.1708.02002`. Accessed: Jan. 31, 2026. [Online]. Available: `http://arxiv.org/abs/1708.02002`.