AI was used lots throughout this project. Almost only for minor things e.g., correcting an error in the code, inspiration on how to code something suggestions to improve performance and so on. As I think AI's don't perform well with big tasks and most of the time it's easier to do it yourself or find a better solution online.
LLMs used: MistralAI, Gemini2.5 flash, Gemini 2.5 pro, Qwen3, a little bit of copilot.

Bigger tasks AI did:

Gemini2.5 Pro:
Input: "Write me a python function, which is a custom learning rate scheduler for a neural network. It should consist of a linear warmup and a cosine decay."
Output:

```
# Define a custom learning rate schedule class with warmup and cosine decay
class WarmupCosineDecay(tf.keras.optimizers.schedules.LearningRateSchedule):
    """
    A custom learning rate schedule that implements a linear warmup
    followed by a cosine decay.
    """
    def __init__(self, peak_lr, warmup_steps, decay_steps, alpha=0.0, name=None):
        super().__init__()
        self.peak_lr = peak_lr
        self.warmup_steps = warmup_steps
        self.decay_steps = decay_steps
        self.alpha = alpha
        self.name = name

    def __call__(self, step):
        with tf.name_scope(self.name or "WarmupCosineDecay"):
            # Ensure step is a float for calculations
            step = tf.cast(step, tf.float32)

            # --- 1. Warmup Phase ---
            # Linearly increase the learning rate from 0 to peak_lr
            warmup_lr = self.peak_lr * (step / self.warmup_steps)

            # --- 2. Cosine Decay Phase ---
            # Define the cosine decay schedule
            cosine_decay_schedule = tf.keras.optimizers.schedules.CosineDecay(
                initial_learning_rate=self.peak_lr,
                decay_steps=self.decay_steps,
                alpha=self.alpha
            )
            # Calculate the learning rate for the decay phase.
            # Note: The 'step' for the cosine part must be relative to its start.
            decay_lr = cosine_decay_schedule(step - self.warmup_steps)

            # --- 3. Choose the correct phase ---
            # Use tf.where to select the learning rate based on the current step
            learning_rate = tf.where(
                step < self.warmup_steps,
                warmup_lr,
                decay_lr
            )
            return learning_rate
```

google jules:
Input: make a gitignore file for this repository. do not remove the direcotries "not_relevant", they are relevant. Remove stuff like ipynbcheckpoints or pycashe

```
Output:
# Ignore Jupyter Notebook checkpoints
.ipynb_checkpoints/

# Ignore Python bytecode
__pycache__/
```

Input: all the irrelevant files except for .tex, .bib, .pdf, .png, .jpg should be ignored too. also ignore gitattributes

```
Output:
I've received the latest code review feedback, and I see a potential contradiction in th

The repository's main purpose seems to be machine learning, and we've identified several

If I follow this instruction literally, all the Python and Notebook source files will be

Could you please clarify? Should I:

Proceed with ignoring all files except .tex, .bib, .pdf, .png, and .jpg?
Also include .py and .ipynb files in the whitelist of files to keep?
Take a different approach to the .gitignore file?
Once I have this clarification, I can finalize the .gitignore file correctly.
```

Input: I meant all the irrelevant files except .tex, .bib, .pdf, .png, and .jpg in the documentation directory.

```
Output: gitignore files were published in a branch "add-gitigrnore-rules"
```