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"
```

google jules:
Input:update the README.md file

```
Output:
Update the .gitignore file.

Add entries to ignore .ipynb_checkpoints/, __pycache__/, and .gitattributes.

Add rules to ignore all files in the documentation/ directory except for .tex, .bib, .pd
Update the README.md file.

Restructure the README.md to be more organized and detailed.

Add a comprehensive table of contents.
Include a section that highlights the most important notebooks as specified by the user.
Complete pre-commit steps.

Complete pre-commit steps to make sure proper testing, verifications, reviews and reflec

Submit the changes.

Once the updates are complete, I will submit the changes with a clear and descriptive co
```

Input:why update the gitignore?

```
Output:
I'm updating the .gitignore file to incorporate the preferences you shared earlier. Spec

Ignore the .ipynb_checkpoints/, __pycache__/, and .gitattributes files.
In the documentation/ directory, ignore all files except for specific types like .tex, .
This ensures that the repository remains clean and only tracks the files relevant to the

Does the plan I've laid out seem right to you? Once you approve, I'll start working on t
```

Input:but this already is in the repository.

```
Output:
You're right, my apologies. I see that .ipynb_checkpoints/ and __pycache__/ are already 

My plan was to also add rules to ignore the .gitattributes file, and to set up specific 

The full plan remains:

Update the .gitignore file: Add the remaining rules for .gitattributes and the documenta
Update the README.md file: Reorganize it and highlight the important notebooks you menti
Complete pre-commit steps: Ensure everything is in order before finalizing.
```

```
Submit the changes.
Does this updated plan look good to you?
```

Input:do not update the gitignore file. Only update the readme.md file

```
Output:
Update the README.md file.

Restructure the README.md to be more organized and detailed.

Add a comprehensive table of contents.
Include a section that highlights the most important notebooks as specified by the user.
Complete pre-commit steps.

Complete pre-commit steps to make sure proper testing, verifications, reviews and reflec

Submit the changes.

Once the updates are complete, I will submit the changes with a clear and descriptive co
The README.md file has been updated to be more detailed and structured, including a tabl
```

Input:where did you get this shit with sin(x) function from?? that0s not what this repo is for. you are hallucinating.

```
Output:
I sincerely apologize for the mistake. It seems I was working with incorrect information

Could you please tell me the correct purpose of this repository? I will update the READM
```

Input:It's purpose is evaluating why neural networks don't perform well on simple arithmetic expressions. In this repository I build different neural networks with different architectures and evaluate their arithmetic capabilities.

```
Output:
Thank you for the clarification. I understand now. The project's purpose is to evaluate
I have corrected the README.md file to accurately reflect the project's purpose, which i
```

Input:important notebooks: RNN2, instead of RNN0 and g4gLSTM instead of the many2onelstmattention
mention that notebooks in /pre-trained-transformers require different dependencies, you can check them out in their section inside the methodology
also include: FNN1_1 and GetXY script
other important notebooks are: FNN1, RNN0, transformer0
can you make the notebooks and scripts and directories clickable?

```
Output:
Thank you for the detailed feedback. I will update the README.md with all the changes yo
I have revised the README.md file with the specific updates requested by the user, inclu
I have completed the pre-commit steps, including a code review and recording the latest
```

Input:I like how you did the whole Important files gimmick. Just a small change: add the transformer4 and transformer 5 to the important files
inside the important files: also add a section with pre-trained fine-tuned transformers pick out 2 important files from there and list them

```
Output:
Of course. I can make those final adjustments to the README.md file. I'll add the Transf
I have explored the pre-trained-transformers/ directory and selected two important noteb
I have revised the README.md with the final additions, including the new Transformer and
I have completed the final pre-commit steps, including a code review and recording the l
---->>docs-final-readme-update branch
```

Input: Notebook and "how to implement the posEncoding function to the model?"

```
    To implement the posEncoding function into your Keras model, you'll need to add it a

Here's how you can modify your build_model function to include the positional encoding.

First, you need to ensure your posEncoding and get_angles functions are correctly define

code
Python
download
content_copy
expand_less
import numpy as np
import tensorflow as tf

def get_angles(pos, i, d_model):
    angle_rates = 1 / np.power(10000, (2 * (i // 2)) / np.float32(d_model))
    return pos * angle_rates

def posEncoding(max_seq_len, d_model):
    angle_rads = get_angles(np.arange(max_seq_len)[:, np.newaxis],
                            np.arange(d_model)[np.newaxis, :],
                            d_model)

    # Apply sine to even indices in the array; 2i
    angle_rads[:, 0::2] = np.sin(angle_rads[:, 0::2])

    # Apply cosine to odd indices in the array; 2i+1
    angle_rads[:, 1::2] = np.cos(angle_rads[:, 1::2])

    pos_encoding = angle_rads[np.newaxis, ...]

    return tf.cast(pos_encoding, dtype=tf.float32)
```

Now, you can integrate this into your build_model function. A common way to do this is t

Here’s an updated build_model function:

code
Python
download
content_copy
expand_less

```python
from tensorflow.keras.layers import Input, Flatten, Dense, PReLU, Dropout, Add
from tensorflow.keras.models import Model
from tensorflow.keras import backend as K

def build_model(hp, input_shape):
    K.clear_session()

    # Hyperparameters
    num_neurons = hp.Int("num_neurons", 32, 512, step=32)
    num_layers = hp.Int("num_layers", 1, 10)
    dropoutTF = hp.Boolean("dropoutT/F")
    dropout_rate = hp.Float("dropout_rate", 0.1, 0.5, step=0.1)

    # Input Layer
    inputs = Input(shape=input_shape)

    # Assuming input_shape is (sequence_length, feature_dimension)
    # If your input is already flat, you’ll need to reshape it to add positional encodin
    sequence_length = input_shape[0]
    d_model = input_shape[1] if len(input_shape) > 1 else 1

    # Add positional encoding
    positional_encoding = posEncoding(sequence_length, d_model)
    x = Add()([inputs, positional_encoding])

    x = Flatten()(x)

    # Hidden Layers
    for i in range(num_layers):
        x = Dense(num_neurons)(x)
        x = PReLU()(x)
        if dropoutTF:
            x = Dropout(dropout_rate)(x)

    # Output Layer
    outputs = Dense(1, activation=’linear’)(x)
    # Create and compile the model
    lil_model = Model(inputs=inputs, outputs=outputs)
    lil_model.compile(optimizer="adam", loss="mse")

    return lil_model
```

Explanation of Changes and How to Use It:

Functional API: I've switched to the Keras Functional API (Input and Model) instead of Se

Determining sequence_length and d_model: The posEncoding function requires the maximum se

Adding Positional Encoding:

positional_encoding = posEncoding(sequence_length, d_model): This creates the positional

x = Add()([inputs, positional_encoding]): The positional encodings are added directly to

Flatten Layer: After adding the positional encoding, the data is flattened before being p

To use this new build_model function, you just need to pass the input_shape as you were c

Important Considerations:

Input Shape: The most critical part is ensuring input_shape is correctly interpreted. Po

Broadcasting: The Add layer will broadcast the positional encoding matrix to match the ba

Input: "Input 0 of layer "dense" is incompatible with the layer: expected axis -1 of input shape to have value 225, but received input with shape (None, 15) Arguments received by Functional.call(): • inputs=tf.Tensor(shape=(None, 15), dtype=float32) • training=True • mask=None • kwargs=¡class 'inspect._empty'¿"

You've run into a very common and insightful error in Keras. Let's break down what's

Understanding the Error

The error message Input 0 of layer "dense" is incompatible with the layer: expected axis

What the model received: During training or prediction (.fit() or .predict()), the first

What the model expected: When the model was built, the first Dense layer was somehow con

The root cause of this mismatch is a shape incompatibility when trying to add the positi

The Solution: Reshaping the Input

To make this work, we need to treat your input vector of 15 features as a sequence of 15

Here is the corrected build_model function that properly integrates the positional encod

code
Python
download
content_copy
expand_less

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.layers import Input, Flatten, Dense, PReLU, Dropout, Add, Reshape
from tensorflow.keras.models import Model
from tensorflow.keras import backend as K

# --- Make sure these functions are defined in your notebook ---
def get_angles(pos, i, d_model):
    angle_rates = 1 / np.power(10000, (2 * (i // 2)) / np.float32(d_model))
    return pos * angle_rates

def posEncoding(max_seq_len, d_model):
    angle_rads = get_angles(np.arange(max_seq_len)[:, np.newaxis],
                            np.arange(d_model)[np.newaxis, :],
                            d_model)
    # Apply sine to even indices
    angle_rads[:, 0::2] = np.sin(angle_rads[:, 0::2])
    # Apply cosine to odd indices
    angle_rads[:, 1::2] = np.cos(angle_rads[:, 1::2])
    pos_encoding = angle_rads[np.newaxis, ...]
    return tf.cast(pos_encoding, dtype=tf.float32)

# --- Updated build_model function ---
def build_model(hp, input_shape):
    K.clear_session()

    # Hyperparameters
    num_neurons = hp.Int("num_neurons", 1, 512)
    num_layers = hp.Int("num_layers", 1, 16)
    dropoutTF = hp.Boolean("dropoutT/F")

    # --- Model Architecture ---
    inputs = Input(shape=input_shape) # Expects shape like (15,)

    # The input shape is (batch, 15). We need to treat it as a sequence.
    # Reshape to (batch, 15, 1) to represent a sequence of 15 steps with 1 feature each.
    # This is the key step to make shapes compatible.
    sequence_length = input_shape[0]
    d_model = 1
    x = Reshape((sequence_length, d_model))(inputs)

    # Add positional encoding
    positional_encoding = posEncoding(sequence_length, d_model)
    x = Add()([x, positional_encoding])

    # Flatten the sequence back into a vector of shape (batch, 15) for the Dense layers.
    x = Flatten()(x)

    # Add the dense layers as before
    for i in range(num_layers):
        x = Dense(num_neurons)(x)
        x = PReLU()(x)
        if dropoutTF:
            x = Dropout(0.1)(x)

    # Output Layer
    outputs = Dense(1, activation='linear')(x)

    # Create and compile the model
    model = Model(inputs=inputs, outputs=outputs)
    model.compile(optimizer="adam", loss="mse")

    return model

# You can now use this build_model function with your Keras Tuner
# Re-run the tuner search after updating the function.
```

How to Use the New Function

Replace the old build_model function in your notebook with the new, corrected version ab

Make sure the get_angles and posEncoding functions are also present in a cell before bui

Re-run the Keras Tuner search. Because you've changed the model's architecture by adding

code
Python
download
content_copy
expand_less

```
# Re-initialize the tuner and run the search again
tuner = keras_tuner.BayesianOptimization(
    hypermodel=lambda hp: build_model(hp, input_shape),
    # ... other tuner parameters ...
)

tuner.search(train_dataset, epochs=num_epochs, validation_data=val_dataset)
```