Developing Time Series Forecasting Models with Generative Artificial Intelligence

Juan Morales-Garcia (jmorales8@ucam.edu)
Universidad Católica San Antonio de Murcia

Antonio Llanes
Universidad Católica San Antonio de Murcia

Francisco Arcas-Tunez
Universidad Católica San Antonio de Murcia

Fernando Terroso-Saenz
Polytechnic University of Cartagena

Research Article

Keywords: Deep Learning, Generative Large Language Models (GLLMs), ChatGPT, Copilot, Time series forecasting

Posted Date: November 20th, 2023

DOI: https://doi.org/10.21203/rs.3.rs-3626057/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.
Developing Time Series Forecasting Models with Generative Artificial Intelligence

Juan Morales-García¹*, Antonio Llanes¹, Francisco Arcas-Túnez¹ and Fernando Terroso-Sáenz²

¹*Computer Science Department, Catholic University of Murcia (UCAM), Av. de los Jerónimos, 135, Murcia, 30107, Murcia, Spain.
²Information and Communications Technologies Department, Technical University of Cartagena, Plaza del Hospital, 1, Cartagena, 30107, Murcia, Spain.

*Corresponding author(s). E-mail(s): jmorales8@ucam.edu; allanes@ucam.edu; farcas@ucam.edu; fernando.terroso@upct.es;

Abstract

Nowadays, Generative Large Language Models (GLLMs) have taken the Artificial Intelligent field by storm. One of the domains where these models have been extensively evaluated is in their role of generators of functional source code for software projects. However, their potential as assistants for writing the code necessary to generate and model machine learning or deep learning architectures has not been fully explored to date. For this reason, this work focuses on evaluating the extent to which different tools based on GLLMs, such as ChatGPT or Copilot, are capable of correctly defining the source code necessary to generate viable predictive models. The use case defined is the forecasting of a time series reporting the inner temperature of a greenhouse. The results show that, while it is true that it possible to obtain good accuracy metrics with simple predictors, the composition of models with complex architectures is still far from the alternative of generating them by human data scientists.

Keywords: Deep Learning, Generative Large Language Models (GLLMs), ChatGPT, Copilot, Time series forecasting
1 Introduction

The creation and evolution of Artificial Intelligence (AI) has been one of the most significant advances in the technology and computer-science fields in the last decades [1]. In recent years, a new wave of innovation in AI has given rise to Generative Large Language Models (GLLMs) which are taking over all fields, such as OpenAI ChatGPT or GitHub Copilot [2]. Due to their capacity to operate through natural language, they are intended to operate as intelligent assistants in a wide range of domains [3].

In this context, the creation of Deep Learning (DL) and Machine Learning (ML) models to solve certain cognitive tasks, such as image recognition, video analysis or timeseries forecasting, was a task traditionally reserved for highly skilled programmers or data scientists who designed the algorithms, implemented their logic and carefully tuned their hyperparameters [4].

Due to the aforementioned capability of GLLMs to operate as assistants in many different fields, the present paper explores the possibility of applying such models to automatically generate the source code to instantiate different types of DL and ML algorithms able to solve particular problems. This way, the resulting DL and ML models are eventually crafted in a no-code way by just generating prompts in natural language. As use case to evaluate such an approach, we intend to compose a set of timeseries predictors to forecast the inner temperature of a smart greenhouse several hours ahead based on well-known frameworks in the Python ecosystem such as Keras¹ or Scikit-learn². For a complete overview, we also consider, as baseline, the manual way of instantiating such AI models by directly coding them.

Consequently, the present work aims to answer the following Research Question (RQ): Can existing Generative Large Language Models (GLLMs) be guided via prompt generation to write valid Python source code able to define the architecture of accurate and functional ML and DL models to forecast temperature time series?

By answering this question, we intend to provide a comprehensive and evidence-based evaluation of the feasibility of using GLLMs as assistants during the model instantiation stage within a data-science project. This way, our work is closely linked to some of the challenges defined in the software engineering field, which is constantly changing [5].

The rest of the paper is organized as follows. Section 2 reviews existing studies regarding the usage of GLLMs in the software development field. Section 3 describes particular GLLM tools used in the study, and the DL and ML models that were generated with them. Section 4 puts forward the dataset to perform the comparison along with the analysis and discussion of the obtained results. Section 5 highlights the conclusions and directions for future works.

2 Related Works

It is possible to find in the existing literature several works that evaluate the suitability of the Generative Large Language Models (GLLMs) from multiple perspectives.

¹https://keras.io/
In that sense, one course of action has focused on evaluating the usability of the source code generated by such models. For example, some studies have analyzed the actual feasibility of Copilot, a LLM-based code generation tool, as a code assistant for Python-based environments [6]. Results show that the code snippets generated by such a tool were, in most of the formulated problems, a good starting point for the developers. However, its usage did not actually save time or increased the success rate of the developers involved in the study. A similar approach is put forward in [7] where authors carried out an study where the 28.7% of the coding problems proposed to by Copilot were correctly solved. In this case, authors made use of EvalPlus a recent framework to evaluate the functional correctness of LLM-generated code, such as [8].

Another line of work has proposed the usage of GLLMs as tools to assist the development of software. As a matter of fact, authors in [9] describe how ChatGPT can be integrated, as a third-party module, in different software-development projects. Another example is GenLine a natural language code synthesis tool that relies on a GLLM to improve the developer experience [10]. Then, an evolution of this work combines GenLine with other tool based on GLLMs to invoke language model prompts as macros in a code editor [11].

Some other works have proposed mechanisms to generate more accurate prompts. As a matter of fact, the authors in [12] firstly identify a palette of software-engineering patterns and then, explore a set of prompt patterns that fit each of these patterns. Beyond code generation, other works have evaluated the suitability of GLLMs as research assistants [13]. As use case in such a work, GLLMs are used to identify non-answers in conference calls. Despite its promising results, the paper also remarks certain risks of such models in aspects like validity, biases or knowledge limitations. A similar approach is put forward in [14] where a set of guidelines to use ChatGPT are defined to improve certain research process, related to the research initiation and data preparation phases.

Concerning the integration of GLLMs as tools for the development of Deep Learning (DL) systems, some works have use these type models to compose validators for fuzzy DL libraries such as Pytorch or TensorFlow [15].

Bearing all the aforementioned works, we can see that GLLMs have been analyzed from multiple perspectives within the software-development field. Nonetheless, there is a lack of studies focusing on the feasibility of generating functional DL and ML models directly from GLLMs and evaluating their actual accuracy. To the best of the authors’ knowledge, this is the first attempt to perform a thoughtful comparison between AI-models synthetically generated and a set of baselines directly developed by data scientists.

3 Materials and Methods

This section shows the Generative AI models used for developing another AI models are described in 3.1 and the AI models used for evaluating our purpose in 3.2. In addition, it is shown how the comparison is to be performed in 3.3.
3.1 Generative Artificial Intelligence models

The GLLMs to be discussed in this document are ChatGPT in its versions 3.5, 4 and 4 code, since it is the generative AI with the greatest impact at present, and Copilot, because it an AI tool that integrates with programming environments to help and facilitate coding tasks for the programmer.

• ChatGPT\textsuperscript{3}: ChatGPT is a new generation language model developed by OpenAI. This model has a vast knowledge base drawn from a wide range of online and offline sources, enabling it to understand and respond consistently to a wide variety of queries and questions. Among the various uses that can be attributed to ChatGPT, automatic code generation is becoming one of the more widespread. In this field ChatGPT is able to automatically generate code at the request of any user describing his requirements, with greater or lesser precision. ChatGPT is able not only to generate code, but also to syntactically or semantically check codes, search for errors in a code, translate between programming languages, etc. Three different versions of ChatGPT have been studied, namely,

– ChatGPT 3.5. This is the general purpose free version, which is not connected to the Internet and provides public access.
– ChatGPT 4, is the general purpose paid version, which are connected to the Internet and requires a monthly subscription to access it.
– ChatGPT 4 code, as well as version 4, is a paid version, which are connected to the Internet and requires a monthly subscription to access it, but this version is focused on code generation. In addition, this version allows you to execute the autogenerated code within the generated prompts.

• Copilot\textsuperscript{4}: GitHub Copilot is an AI assistant for writing code. It is a natural language model based on OpenAI Codex, a modified version of GPT-3. The Codex model is trained on gigabytes of source code in a dozen programming languages such as Python, JavaScript, TypeScript, Ruby or Go among others. Its integration with IDEs such as Visual Studio Code, Visual Studio or JetBrains allows for real-time code hints while writing code. GitHub Copilot allows to generate code solutions from statements expressed in natural language. It can also provide explanations of code and translate code between different programming languages, as well as provide auto-completion suggestions, create unit tests or detect security vulnerabilities in the code.

3.2 Artificial Intelligence models

To assess the impact on the accuracy of the AI models developed by GLLMs versus the AI models developed by an expert, first it is necessary to explain the AI models to be developed. These AI models have been separated into Machine Learning models (see 3.2.1) and Deep Learning models (see 3.2.2).

\textsuperscript{3}https://openai.com/chatgpt
\textsuperscript{4}https://github.com/features/copilot
3.2.1 Machine Learning models

The Machine Learning models used in this study are:

- **Random Forest [16] Regressor (RFR):** Is an improvement of Regression Decision Trees (DTR), which provide model trees that are simple to convert into decision rules, making them intuitive machine learning models. DTRs have the issue that they frequently suffer from overfitting, hence several decision trees constructed randomly are commonly employed along with a decision system that allows the model to be improved. However, this strategy may not be adequate for the model to learn the properties of the model. This particular collection of trees makes up the Random Forest algorithm because it resembles a forest of arbitrary trees. Together with ANNs, this technique produces excellent results in regression and is popular because of its speed and robustness.

- **K-Nearest Neighbors [17] Regressor (KNNR):** This algorithm assigns a value to a new sample depending on how similar it is to the samples in the training set. It is based on the similarity of sample attributes. The distance between each training point and the new point is first computed. The distance has been estimated for this project using the Euclidean distance calculation, but it can also be determined using the Manhattan distance formula. Although it is also employed in regression issues, the KNNR technique is rather more common for classification problems. These approaches have the drawback of requiring a specified number of neighbors (K) to be taken into account in relation to the new sample. The choice of K’s value is crucial because if it is too small, overfitting, or a classification that is too similar to the training set, may occur. On the other hand, an excessively high value will result in an untrained model.

3.2.2 Deep Learning models

The Deep Learning models used in this study are the following:

- **MultiLayer Perceptron [18] (MLP):** It is a model that imitates how a group of biological neurons function. Although it is more frequently used in classification, it can also be used in regression models. A single perceptron (or artificial neuron) can be compared to a logistic regression. The multilayer perceptron (MLP), which is the structural unit of the Artificial Neural Network, or ANN, is made up of several perceptrons in each layer. The input, hidden, and output layers make up the three layers of the MLP that we employ in this work. The hidden layer processes the inputs, the input layer receives the input features, and the output layer generates the output. In essence, every layer tries to learn specific weights. Due to artificial neural networks use activation functions to recognize non-linear network properties, they can learn any intricate relationship between input and output.

- **Convolutional Neural Network [19] (CNN):** It is a model that is employed in a variety of contexts and fields, but they are most frequently used in imaging for classification. However, they are also used in regression, where they can be applied to time series data by transforming the information to make it compatible with the convolutional network’s inputs. A CNN is made up of blocks of filters that
enable the relevant features to be extracted from the input through convolutional operations. The automatic learning of the filters that allows the necessary and most relevant features to be extracted from the input data is one of the advantages of CNNs over traditional neural networks (ANNs).

* Long Short-Term Memory [20] (LSTM): it is a widely used model of Deep Learning due to it has the advantage of allowing for long-term memory in addition to working with time series like recurrent models. A LSTM is a subtype of recurrent neural architecture with a state memory and multilayer cell structure. An LSTM unit is made up of a cell, an input gate, an output gate, and a forget gate. The three gates control the flow of data into and out of the cell, and the cell remembers values over arbitrary time intervals. The LSTM differs from a traditional recurrent network in that it can choose whether to keep the existing memory through the use of doors instead of overwriting it at each time step. In order to detect long-distance dependencies, the LSTM unit must first identify an important characteristic of an input sequence early in the process.

3.3 Comparison Methodology

In this section we describe the methodology used in the different coding sections, there being practically only differences between coding by the programmer and the rest of the generative artificial intelligences, since the methodology between the different generative artificial intelligences only differs in the execution, because conceptually it has been the same.

With respect to the coding of the algorithms by a programmer, it has been the usual way of working, coding from the programmer’s own knowledge, until the results obtained are achieved. Selecting also the hyperparameters carefully by means of repeated comparisons until no more improvements were obtained.

Regarding ChatGPT, the methodology followed to achieve the automatic code generation has been to do not manually modify the code generated by ChatGPT under any circumstance, to check if through a single prompt or correcting the generated errors it would be possible to obtain a 100% functional code. In certain cases, we have re-written the prompt in order to verify that without programming expertise we could reach valid results. The prompts that have generated the code for the different algorithms are presented below:

* Considering that we have a csv with a temperature time series with data taken every 60 minutes in the column called MeasurementBox1.TBS. (file name DS-CLEAN-60.csv). When reading the file, the file uses separators ”,”, the decimal point as ”.”, the column names are in the first row of the file, and the first column is the column that is used as the index. Can you make a "RFR/KNR/MLP/CNN/LSTM" algorithm to predict the next 24 temperatures? By preparing test data for windows of days (each day is 24 data). For a prediction you should base on the previous 24 data, using previous predictions, but not the original values. I want you to print on screen only the 24 temperature predictions (one per line please), and as quality measures print the coefficient of determination, MAE, RMSE, MAPE and CVRMSE.
The above prompt has been the starting point in each GLLM to generate an instance of each of the models listed in 3.2. Next, subsequent modifications have been derived from errors or misunderstandings in each GLLM setting. As shown, the prompt is divided into different parts. First, it tells which paths and filenames to use. Then, it gives some guidelines to be able to correctly process the data file. Finally, we define the particular algorithm we want to generate. Besides, we define how and from what data we want it to generate the new predictions, and how we want it to display the results. Finally, we also indicate the metrics we want to collect to validate the results.

Regarding Copilot, the methodology has been quite similar, we have used Visual Studio as framework, with the Copilot plugin\(^5\), in which it is possible to describe as a comment what we want to do (basically the aforementioned prompt). As outcome, the plugin generates the code, which we did not manually modify as in the case of ChatGPT.

We want to emphasize that the main objective with GLLMs was not to hyper-parameterize the models, not even to find the best values for the parameters of the models, nor to make these values similar to those chosen by the programmer. Bearing in mind the formulated RQ, the main goal during the models generation has been to check if it was able to generate functional models without no intervention of a programmer, only through a simple prompt.

### 4 Evaluation and Discussion

The following is an Exploratory Data Analysis (see 4.1), the running environment in which all tests have been launched (see 4.2) and the evaluation of each developed model (see 4.3).

#### 4.1 Dataset and Exploratory Data Analysis

In order to carry out the proposed study, time series temperature data were collected inside a precision agriculture greenhouse installed in the area of the Region of Murcia (southeastern Spain) between 2022-07-01 and 2023-07-01. The data come from a sensor installed in a small weather station located inside the greenhouse. The data are collected with a periodicity of 5 minutes, approximately. As 5 minutes is a very small temporally that is not useful for decision making, the temporality has been extended to 60 minutes by aggregating the original data.

Table 1 shows the statistics of the dataset used in this study. More specifically, the total sample instances (count), mean, standard deviation (std), minimum value (min), maximum value (max), quartiles (25%, 50% and 75%) and maximum value (max) are shown.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>count</th>
<th>mean</th>
<th>std</th>
<th>min</th>
<th>25%</th>
<th>50%</th>
<th>75%</th>
<th>max</th>
</tr>
</thead>
</table>

*Table 1: Dataset statistics.*

\(^5\)https://marketplace.visualstudio.com/items?itemName=GitHub.copilot
As can be seen, the variable under study (IndoorTemperature), has: a minimum value of 10.197 °C; a maximum value of 33.383 °C; a mean of 21.639 °C with a standard deviation of 5.226 °C.

Figure 1 shows an Exploratory Data Analysis of the dataset used in this study. More specifically, Figure 1a shows the autocorrelation function, which indicates how often (lags) a cycle is created. On the other hand, Figure 1b shows the box-and-whisker plot that allows us to visualize the distribution of the data: its maximum and minimum values, possible outliers, quartiles and median. Finally, Figure 1c shows the seasonal decomposition of the time series under study, which shows: (1) the time series, (2) its trend, (3) its seasonality and (4) its residual (noise).

As can be seen, the variable under study (IndoorTemperature), has a daily temporality (every 96 values), with a stable trend over time, a well-defined seasonal component and little noise in the time series, which makes it a good target to forecast.

### 4.2 Running environment

In order to run all the tests presented in this manuscript, we have made use of a server (named ‘mercurio’) with the following hardware characteristics: An Intel(R)
Xeon(R) Gold 6226R CPU, with 16 cores at 2.90GHz, 196 GigaBytes DDR4 2933 MHz of RAM memory and 22 MB of cache memory. 2x Quadro RTX 5000 16GB GDDR6 384 Tensor cores, 3072 CUDA cores and NVLink PCI Express x16 3.0. A Solid State Drive with 15 TeraBytes. Regarding de software characteristics, we must mention that mercurio is on a Ubuntu 20.04 LTS operative system, and we used to implement Python 3.8 with TensorFlow and keras in version 2.12 over a Jupyter Notebook.

4.3 Model evaluation

Within this subsection, the following is shown a brief description of the metrics used to evaluate the accuracy of the AI models (see 4.3.1), all hyperparameters used as well as a brief description of them (see 4.3.2) and, finally, the results obtained for each of the AI models developed when trying to forecast next day’s temperature every 60 minutes (see 4.3.3).

4.3.1 Metrics

In order to properly evaluate the prediction quality of the implemented Deep Learning models, five different metrics have been applied: (1) $R^2$, (2) RMSE, (3) MAE, (4) MAPE and (5) CVRMSE.

1. $R^2$: Coefficient of determination (R Squared) is a metric that indicates the proportion of the variance for a dependent variable that’s explained by an independent variable, whose formula is defined as: $R^2(y, \hat{y}) = \frac{\sum_{i=1}^{n}(y_i - \bar{y})(\hat{y}_i - \bar{\hat{y}})}{\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2 \sum_{i=1}^{n}(\hat{y}_i - \bar{\hat{y}})^2}}$

2. RMSE: Root Mean Squared Error is a metric that indicates what is the level of dispersion of these residual values, whose formula is defined as: $\text{RMSE}(y, \hat{y}) = \sqrt{\frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{N}}$

3. MAE: Mean Absolute Error is a metric that indicates the mean difference between the predicted value and the actual value at each predicted point, whose formula is defined as: $\text{MAE}(y, \hat{y}) = \frac{\sum_{i=0}^{N-1}|y_i - \hat{y}_i|}{N}$

4. MAPE: Mean Absolute Percentage Error is a metric that indicates the size of the (absolute) error in percentage terms, whose formula is defined as: $\text{MAPE}(y, \hat{y}) = \frac{100\%}{N} \sum_{i=0}^{N-1} \frac{|y_i - \hat{y}_i|}{y_i}$

5. CVRMSE: Coefficient of the Variation of the Root Mean Square Error is a metric that indicates instability in the observed relationship between variables in the baseline period, whose formula is defined as: $\text{CVRMSE}(y, \hat{y}) = \frac{\text{RMSE}(y, \hat{y})}{\bar{y}}$

All of the previously described metrics are given two parameters: (1) $y$, the vector of ground-truth values; and (2) $\hat{y}$, the vector of forecasted values by the AI models.

4.3.2 Hyperparametrization

In order to properly adapt Artificial Intelligence models to the study scenario, it is necessary to perform hyperparameter tuning for the models. A brief description of each of the hyperparameters used is provided below:
• **Jobs**: Number of workers (threads or processes) that are spawned in parallel.
• **Estimators**: Number of trees created in a ensemble of forests.
• **Neighbors**: Number of neighbors used to forecast a result.
• **Units**: Number of neurons used in hidden layers.
• **Filters**: Features detector.
• **Kernel size**: Filters matrix used to extract the features from the dataset.
• **Strides**: Number of pixels shifts over the input matrix.
• **Activation function**: Function that decides if a neuron should be (or not) activated.
• **Batch size**: Size of bach used for training/forecasting.
• **Epochs**: Number of epoch used in training.
• **Optimizer**: Function that optimizes the learning of an artificial intelligence model, updating its neurons’ weights depending on the error evaluation.
• **Loss function**: Function used to evaluate the error of the model in each epoch.
• **Learning rate**: Percentage change with which weights are updated at each iteration.

All hyperparameters used in each benchmark are shown in the table 2.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>ChatGPT 3.5</th>
<th>ChatGPT 4</th>
<th>ChatGPT 4 Code</th>
<th>Copilot</th>
<th>Programmer</th>
</tr>
</thead>
<tbody>
<tr>
<td>RFR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jobs</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>Estimators</td>
<td>100</td>
<td>100</td>
<td>500</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>Neighbors</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>96</td>
</tr>
<tr>
<td>Units</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>70</td>
<td></td>
</tr>
<tr>
<td>Activation function</td>
<td>ReLU</td>
<td>ReLU</td>
<td>ReLU</td>
<td>ReLU</td>
<td>tanh</td>
</tr>
<tr>
<td>Batch size</td>
<td>32</td>
<td>32</td>
<td>Auto</td>
<td>Auto</td>
<td>2880</td>
</tr>
<tr>
<td>Epochs</td>
<td>100</td>
<td>100</td>
<td>500</td>
<td>100</td>
<td>3000 (+ Early Stopping)</td>
</tr>
<tr>
<td>Optimizer</td>
<td>Adam</td>
<td>Adam</td>
<td>Adam</td>
<td>Adam</td>
<td>Adam</td>
</tr>
<tr>
<td>Loss function</td>
<td>MSE</td>
<td>MSE</td>
<td>MSE</td>
<td>MSE</td>
<td>MSE</td>
</tr>
<tr>
<td>Learning rate</td>
<td>0.001</td>
<td>0.001</td>
<td>0.001</td>
<td>0.001</td>
<td>0.003 (+ ReduceLROnPlateau)</td>
</tr>
<tr>
<td>Filters</td>
<td>32</td>
<td>32</td>
<td>32</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>Kernel size</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Strides</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Activation function</td>
<td>ReLU</td>
<td>ReLU</td>
<td>ReLU</td>
<td>ReLU</td>
<td>tanh</td>
</tr>
<tr>
<td>Batch size</td>
<td>16</td>
<td>16</td>
<td>100</td>
<td>100</td>
<td>2880</td>
</tr>
<tr>
<td>Epochs</td>
<td>50</td>
<td>50</td>
<td>100</td>
<td>100</td>
<td>3000 (+ Early Stopping)</td>
</tr>
<tr>
<td>Optimizer</td>
<td>Adam</td>
<td>Adam</td>
<td>Adam</td>
<td>Adam</td>
<td>Adam</td>
</tr>
<tr>
<td>Loss function</td>
<td>MSE</td>
<td>MSE</td>
<td>MSE</td>
<td>MSE</td>
<td>MSE</td>
</tr>
<tr>
<td>Learning rate</td>
<td>0.001</td>
<td>0.001</td>
<td>0.001</td>
<td>0.001</td>
<td>0.003 (+ ReduceLROnPlateau)</td>
</tr>
<tr>
<td>Units</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>70</td>
<td></td>
</tr>
<tr>
<td>Activation function</td>
<td>ReLU</td>
<td>ReLU</td>
<td>ReLU</td>
<td>ReLU</td>
<td>tanh</td>
</tr>
<tr>
<td>Batch size</td>
<td>16</td>
<td>16</td>
<td>100</td>
<td>100</td>
<td>2880</td>
</tr>
<tr>
<td>Epochs</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>3000 (+ Early Stopping)</td>
</tr>
<tr>
<td>Optimizer</td>
<td>Adam</td>
<td>Adam</td>
<td>Adam</td>
<td>Adam</td>
<td>Adam</td>
</tr>
<tr>
<td>Loss function</td>
<td>MSE</td>
<td>MSE</td>
<td>MSE</td>
<td>MSE</td>
<td>MSE</td>
</tr>
<tr>
<td>Learning rate</td>
<td>0.001</td>
<td>0.001</td>
<td>0.001</td>
<td>0.001</td>
<td>0.003 (+ ReduceLROnPlateau)</td>
</tr>
</tbody>
</table>

**Table 2**: Hyperparameters used for each model.
### 4.3.3 Model results

<table>
<thead>
<tr>
<th>Source</th>
<th>Metrics</th>
<th>$R^2$</th>
<th>RMSE</th>
<th>MAE</th>
<th>MAPE</th>
<th>CVRMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChatGPT 3.5</td>
<td>RFR</td>
<td>0.68</td>
<td>6.23</td>
<td>5.26</td>
<td>20.88%</td>
<td>25.44%</td>
</tr>
<tr>
<td></td>
<td>KNNR</td>
<td>0.73</td>
<td>2.48</td>
<td>2.02</td>
<td>18.98%</td>
<td>10.15%</td>
</tr>
<tr>
<td></td>
<td>MLP</td>
<td>0.48</td>
<td>3.35</td>
<td>2.57</td>
<td>12.53%</td>
<td>15.74%</td>
</tr>
<tr>
<td></td>
<td>CNN</td>
<td>0.73</td>
<td>4.24</td>
<td>3.56</td>
<td>15.20%</td>
<td>17.93%</td>
</tr>
<tr>
<td></td>
<td>LSTM</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>RFR</td>
<td>0.71</td>
<td>4.36</td>
<td>3.90</td>
<td>19.90%</td>
<td>23.34%</td>
</tr>
<tr>
<td>ChatGPT 4</td>
<td>KNNR</td>
<td>0.98</td>
<td>1.74</td>
<td>1.50</td>
<td>18.09%</td>
<td>9.31%</td>
</tr>
<tr>
<td></td>
<td>MLP</td>
<td>0.64</td>
<td>2.34</td>
<td>1.91</td>
<td>11.94%</td>
<td>14.44%</td>
</tr>
<tr>
<td></td>
<td>CNN</td>
<td>0.98</td>
<td>2.97</td>
<td>2.64</td>
<td>14.49%</td>
<td>16.45%</td>
</tr>
<tr>
<td></td>
<td>LSTM</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>RFR</td>
<td>0.71</td>
<td>4.33</td>
<td>3.82</td>
<td>17.65%</td>
<td>21.53%</td>
</tr>
<tr>
<td>ChatGPT 4 code</td>
<td>KNNR</td>
<td>0.76</td>
<td>1.72</td>
<td>1.47</td>
<td>16.04%</td>
<td>8.59%</td>
</tr>
<tr>
<td></td>
<td>MLP</td>
<td>0.50</td>
<td>2.33</td>
<td>1.87</td>
<td>10.59%</td>
<td>13.32%</td>
</tr>
<tr>
<td></td>
<td>CNN</td>
<td>0.99</td>
<td>1.87</td>
<td>1.68</td>
<td>7.21%</td>
<td>7.89%</td>
</tr>
<tr>
<td></td>
<td>LSTM</td>
<td>0.76</td>
<td>2.95</td>
<td>2.59</td>
<td>12.85%</td>
<td>15.17%</td>
</tr>
<tr>
<td></td>
<td>RFR</td>
<td>0.69</td>
<td>4.32</td>
<td>3.68</td>
<td>17.63%</td>
<td>21.44%</td>
</tr>
<tr>
<td>Copilot</td>
<td>KNNR</td>
<td>0.74</td>
<td>1.72</td>
<td>1.41</td>
<td>16.03%</td>
<td>8.55%</td>
</tr>
<tr>
<td></td>
<td>MLP</td>
<td>0.49</td>
<td>2.32</td>
<td>1.80</td>
<td>10.58%</td>
<td>13.27%</td>
</tr>
<tr>
<td></td>
<td>CNN</td>
<td>0.94</td>
<td>1.83</td>
<td>1.72</td>
<td>7.78%</td>
<td>7.91%</td>
</tr>
<tr>
<td></td>
<td>LSTM</td>
<td>0.74</td>
<td>2.94</td>
<td>2.49</td>
<td>12.83%</td>
<td>15.11%</td>
</tr>
<tr>
<td></td>
<td>RFR</td>
<td>0.80</td>
<td>5.43</td>
<td>4.82</td>
<td>18.83%</td>
<td>22.18%</td>
</tr>
<tr>
<td>Programmer</td>
<td>KNNR</td>
<td>0.82</td>
<td>3.89</td>
<td>3.16</td>
<td>11.96%</td>
<td>19.92%</td>
</tr>
<tr>
<td></td>
<td>MLP</td>
<td>0.91</td>
<td>2.36</td>
<td>2.02</td>
<td>9.19%</td>
<td>9.64%</td>
</tr>
<tr>
<td></td>
<td>CNN</td>
<td>0.92</td>
<td>1.78</td>
<td>1.59</td>
<td>6.89%</td>
<td>7.30%</td>
</tr>
<tr>
<td></td>
<td>LSTM</td>
<td>0.86</td>
<td>2.82</td>
<td>2.38</td>
<td>10.83%</td>
<td>11.54%</td>
</tr>
</tbody>
</table>

**Table 3:** Evaluation results obtained for each of the generated models. The best value for each metric and model is shown in bold.
Table 3 shows the obtained metrics and it provides some interesting results. Focusing on the models by the generative AI tools, Copilot was the one that gave rise to the most accurate predictors according to most metrics. It is also true that the models generated by ChatGPT 4 code also obtained quite promising results. However, the version 4 and 3.5 of ChatGPT generated models with a lower accuracy. For instance, the CVRSME of the RFR designed by these two frameworks was 25.44\% and 23.34\% and the one by Copilot was 21.44.

Concerning human-generated instances, we can see that the manually-crafted DL models, outperformed the ones suggested by the generative-AI sources. As matter of fact, the values in bold in Table 3 indicate that the human-generated MLP, CNN and LSTM instances obtained the best values in almost all of the metrics. However, the ML instances exhibited a different pattern. In particular, RFR and KNNR models achieved slightly better results when they were synthetically generated. For example, the RFR by Copilot obtained lower values than the human-based version for RMSE (4.32 vs 5.43), MAE (3.68 v 4.82) and CVRMSE (21.44 vs 22.18). A similar pattern is shown for the KNKR model.

These results show that the models defined by generative AI did not performed well when the required complexity of the target model grew. This suggests that the development of AI models whose architecture involves the definition and tuning of a large number of parameters, such as the DL models used as baselines in the evaluation, still requires a human factor at least in the definition and design of a model’s architecture.

All in all, we can now answer the RQ formulated in sec. 1, Can existing Generative Large Language Models (GLLMs) be guided via prompt generation to write valid Python source code able to define the architecture of accurate and functional ML and DL models to forecast temperature time series? Yes, they can generate functional and valid Python code to instantiate a large palette of timeseries predictors based on ML and DL models. However, the accuracy of such instances is still lower than the one achieved by alternative instances directly generated by humans. This is specially remarkable in the case of the DL models where, for example, the LSTM generated by data-scientists obtained the best values for the 5 metrics under consideration.

5 Conclusion and Future Work

The use of Generative Large Language Models (GLLMs) in the software engineering field is a promising area of research. Some studies have already evaluated GLLMs as assistants for generating valid source code in different software development settings. However, the actual application of these tools to assist developers and data scientists in designing the architecture of machine learning (ML) or deep learning (DL) instances within a data science development project has not been fully explored in the literature.

This paper investigates the possibility of generating functional source code to compose time series predictors based on ML and DL algorithms without requiring
advanced programming or data science skills. Using a palette of models and evaluation metrics and applying multiple GLLM-based tools, the results demonstrate that generative AI can indeed write source code for functional models. The evaluation study showed that the accuracy of these "synthetic" instances was lower compared to those "manually" generated by data scientists, especially for DL algorithms whose architecture involves multiple dimensions and parameters. Nevertheless, it is also true that the ML instances generated with GLLMs achieved results quite close to those obtained by the human-generated models.

The results presented in this work could have a significant impact on multiple fields, such as the development of software products that require the inclusion of lightweight and simple AI components or the educational sector where school children and high school students could use and experiment with functional ML models without coding them beforehand.

As future work, a detailed qualitative study of the influence of prompts on code generation will be conducted. In this regard, we will investigate whether it is possible to perform the hyperparameter configuration step through prompt generation. This way, we will enable GLLMs to generate not only the code snippets comprising the architecture of the models but also a tentative configuration of certain hyperparameters to train them.

**Declarations**

**Ethical Approval**

Not applicable.

**Conflict of interest**

The authors declare that they have no conflict of interest.

**Authors’ contributions**


**Consent to publish**

All authors have read and agreed to the published version of the manuscript.

**Funding**

This work is derived from R&D projects RTC2019-007159-5, PID2020-112827GB-I00 and CD/AG/MI3, funded by MCIN/AEI/10.13039/501100011033, “FSE invest in your future”.
Availability of data and materials

All data and materials are available on request from the authors of this paper.

References


ChatGPT: Applications, Techniques, and Implications, pp. 35–51. Springer, ???. (2023)


[13] de Kok, T.: Generative llms and textual analysis in accounting:(chat) gpt as research assistant? Available at SSRN (2023)


