Implementacija jezičnog asistenta za statističke analize korištenjem velikih jezičnih modela

Krišković, Filip

UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

MASTER THESIS No. 478

IMPLEMENTATION OF A LANGUAGE ASSISTANT FOR STATISTICAL ANALYSIS USING LARGE LANGUAGE MODELS

Filip Krišković

Zagreb, June 2024

UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

MASTER THESIS No. 478

IMPLEMENTATION OF A LANGUAGE ASSISTANT FOR STATISTICAL ANALYSIS USING LARGE LANGUAGE MODELS

Filip Krišković

Zagreb, June 2024

UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

Zagreb, 04 March 2024

MASTER THESIS ASSIGNMENT No. 478

Student:	**Filip Krišković (0036523534)**
Study:	Computing
Profile:	Computer Science
Mentor:	assoc. prof. Marina Bagić Babac

Title: **Implementation of a language assistant for statistical analysis using large language models**

Description:

This thesis investigates the need and process of creating a language assistant that provides answers to user queries related to CSV files. User queries are transformed from natural to programming language in order to make it possible to perform statistical calculations on the entered data and provide the user with answers in natural language. To implement the assistant, large open source language models should be used, adapted to the user's requirements using deep learning techniques and natural language processing. The language assistant's performance will be evaluated using appropriate metrics in order to assess its effectiveness and reliability in providing answers to the questions posed.

Submission date: 28 June 2024

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

DIPLOMSKI ZADATAK br. 478

Pristupnik:	**Filip Krišković (0036523534)**
Studij:	Računarstvo
Profil:	Računarska znanost
Mentorica:	izv. prof. dr. sc. Marina Bagić Babac

Zadatak: **Implementacija jezičnog asistenta za statističke analize korištenjem velikih jezičnih modela**

Opis zadatka:

Ovaj diplomski rad istražuje potrebu i proces izrade jezičnog asistenta koji pruža odgovore na upite korisnika vezane uz CSV datoteke. Korisnički upiti se transformiraju iz prirodnog u programski jezik kako bi se omogućilo izvođenje statističkih izračuna nad unesenim podacima te korisniku pružili odgovori u prirodnom jeziku. Za implementaciju asistenta treba koristiti velike jezične modele otvorenog koda, prilagođene zahtjevima korisnika pomoću tehnika dubokog učenja i obrade prirodnog jezika. Odgovarajućim metrikama evaluirati performanse jezičnog asistenta kako bi se procijenila njegova učinkovitost i pouzdanost u pružanju odgovora na postavljena pitanja.

Rok za predaju rada: 28. lipnja 2024.

Table of Contents

1. Introduction

In today's age of fast improving technologies, the ability to derive meaningful conclusions from vast datasets is more important than ever. To get the most out of the available resources is paramount for growth and development, and statistical analysis or artificial intelligence can significantly help with such information extraction. However, due to the complexity of traditional statistical tools, programming languages and the knowledge required to use artificial intelligence, to most people, extracting new data from datasets poses a challenge. As a result, there exists a need for an intuitive and user-friendly solution that can seamlessly interact with the data, transcending the intricacies of code and syntax. With this in mind, this paper delves into the development and implementation of a language assistant capable of processing natural language and doing statistical analysis by leveraging the power of large language models. Large language models have significantly improved human-computer interaction due to the advances in natural language processing. These models are capable of understanding and generating natural language not because of their complexity, but rather the vast amount of data that was used for their training. Such models can be applied to a variety of general tasks or even fine-tuned to execute a more specific request more accurately. The inspiration for this paper came from [1], where they created an autonomous GIS system, powered by AI. Their implementation used gpt-4, and later updated with gpt-4o, which isn't available to free OpenAI users. Therefore this thesis focuses on using the power of free open-source models to create a whole language assistant that excels at interpreting user queries expressed in natural language and translating them into executable code for statistical analysis. The focus of this research lies in providing users with an intuitive interface to interact with data in CSV (Comma-Separated Values) format, a ubiquitous data interchange format commonly used for storing tabular data and that is accessible to everyone. This assistant serves to bridge the gap between human communication and data analytics, without the need for users to know any statistical analytics or programming languages. By understanding and processing queries in natural language, the assistant enables users to perform a wide array of statistical calculations on their data with ease. By translating user queries into programming language constructs, the assistant facilitates the execution of statistical operations on the provided dataset, subsequently delivering the results in a comprehensible and accessible manner. Lastly, the models that are used as the brain of the system are evaluated based on the understanding of the natural language as well as the accuracy of the generated code, it's execution and the provided result. The ultimate goal of this paper is to contribute to the ongoing development surrounding the fusion of natural language processing and data analytics and making statistical analysis more accessible and more usable.

2. Related work

As mentioned, this research paper was inspired by the autonomous GIS paper by [1], in it, the authors proposed that an autonomous GIS "requires five critical modules, including decision-making (LLM as the core), data collecting, data operating, operation logging, and history retrieval" [1]. These modules are crucial for any GIS to achieve "five autonomous goals: self-generating, self-organizing, self-verifying, self-executing, and self-growing" [1].

Autonomous Goal	Involving Modules	Functionality
Self-generating	Decision-making, Data collecting	Generate solutions and data operation programs
Self-organizing	Decision-making, Operation logging	Ensure the operations are executed in the correct order, and data is stored in an appropriate manner
Self-verifying	Decision-making, Data operating	Test and verify the generated workflow, code, and programs
Self-executing	Data operating, Operation logging	Execute generated workflows, code, or programs
Self-growing	Operation logging, History retrieval	Reuse the verified operations

Figure 1, Autonomous goals and modules of autonomous GIS [1]

In their prototype, the LLM-Geo, they implemented decision-making and data operating which ensured three autonomous goals, i.e., self-generating, self-organizing, and self-executing, with other modules in development. This paper also follows this approach and contains both solution generation, as well as solution execution, with additional checks. The following flowchart shows the process that LLM-Geo goes through when answering user queries.

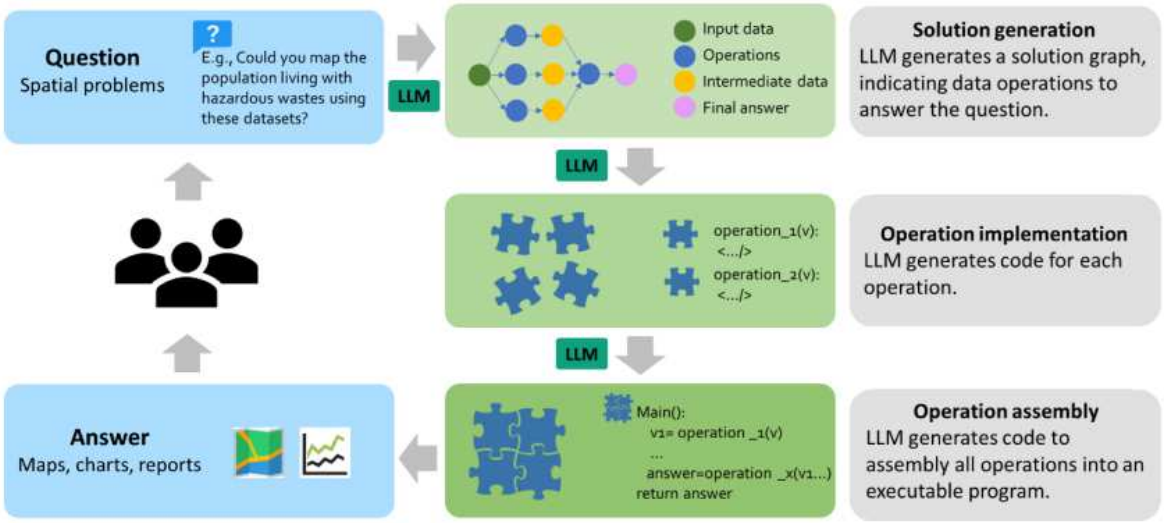


Figure 2, The overall workflow of LLM-Geo

"The process begins with the user inputting the spatial question along with associated data locations" [1], these can be URLs, REST services, APIs or even local paths. After that, the LLM generates a solution in a form of a directed acyclic graph that contains operation nodes and data nodes. "A data node refers to an operation's input data or output data, consisting of three types: input data node, intermediate data node, and output data node" [1]. "An operation node is a process to manipulate data with input and output. Its inputs can be the input data nodes or intermediate data nodes from the ancestor operation nodes" [1]. Based on this, the LLM gets the necessary information about each operation node, to begin the operation implementation step. Per operation node, this step consists of gathering the nodes that precede and follow the current operation node, as well as the original user question, the data sources, and the whole graph from the previous step. "The current version of LLM-Geo generates a Python function for each operation; the function definition and return data (i.e., names of functions and function

input/output variables) are pre-defined in the solution graph. This strategy is used to reduce the uncertainty of code generation" [1]. After each operation was generated, LLM-Geo gathers them and delivers them to the Operation assembly, where the LLM generates an assembly code that connects all of the operations together into a final solution that is then executed and provided back to the user.

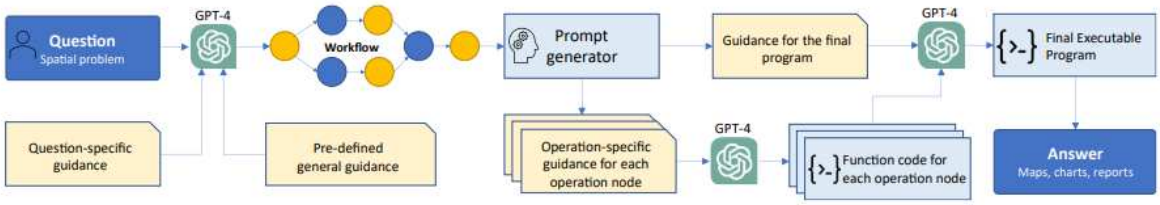


Figure 3, Implementation workflow of LLM-Geo with GPT-4 API [1]

"Human problem-solving often employs a divide-and-conquer strategy when facing complex tasks. By adopting this approach in the design of LLM-Geo, it is intended to address spatial analysis tasks by breaking complex problems into smaller, more manageable sub-problems that LLMs can handle" [1]. Even though the test examples that they displayed in their paper are straightforward, in a sense that a model like gpt-4o can generate a correct solution straight away, for more complex question it would be more beneficial for the LLM itself to first make a plan of action and then implement it, so that it would yield better results.

An effective technique for "enhancing the accuracy and reliability of generative AI models with facts fetched from external sources" [2] is Retrieval augmented generation (RAG). This is what fills the gaps of how LLMs work and give them a deep understanding of the knowledge inside of data, also known as parameterized knowledge [2]. Same as with this paper's system, RAG allows users to, relatively easily, "have a conversation with data repositories" [2]. The way RAG works is by building a vector knowledge base and converting a user query into an embedding vector that can essentially be cross-referenced and searched in the vector database. This allows for a fast and reliable way of looking up information that way already verified, processed and stored. Pairing this with an LLM, instead of having an LLM search the web for answers through an API, RAG allows a faster fetching of data that is more relevant to the existing role of a system it is implemented in. With the retrieved information, an LLM is then able to simply form an answer, and present it to the user, and potentially even citing the sources the information was obtained from [2].

Additionally, "the embedding model continuously creates and updates machine-readable indices – vector databases, for new and updated knowledge bases as they become available" [2].

Here is a simplified view of how RAG and LLMs are combined to provide more accurate information with their answers:

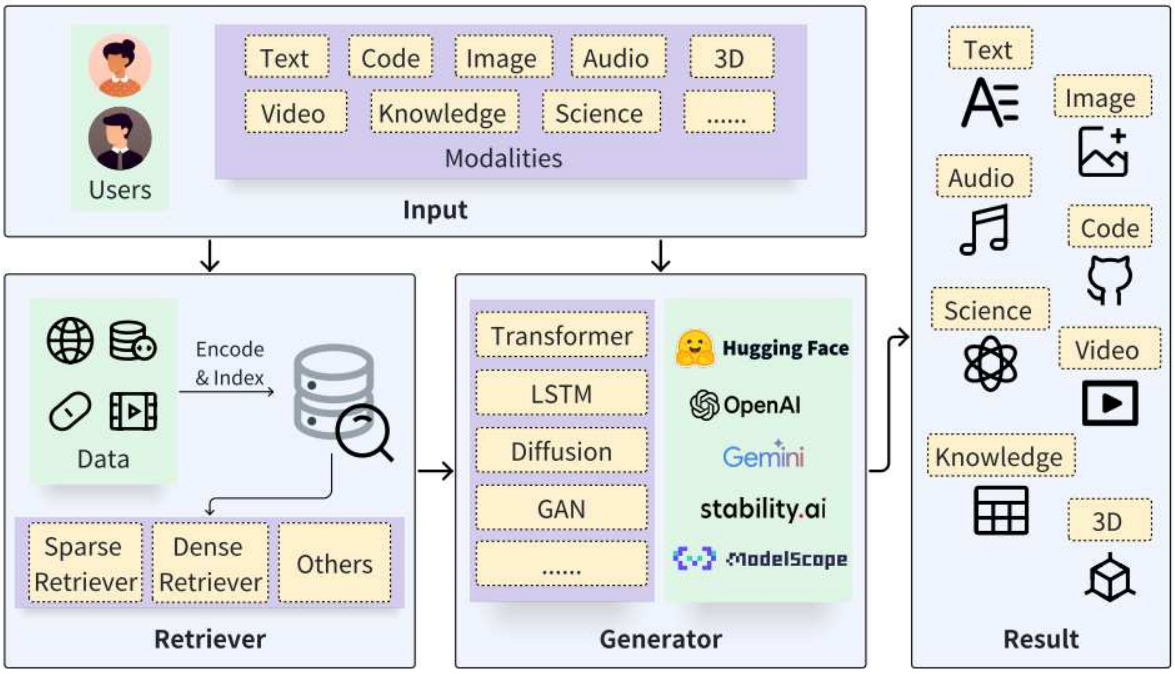


Figure 4, a generic RAG architecture [3]

"The quality of retrieved content, in RAG systems, determines the information that is fed into the generators" [3]. A lower quality of the information degenerates the system to the hallucination-prone one, that does not use RAG at all. [3] introduce some effective methods to enhance the effectiveness of information retrieval. These include recursive retrieval, chunk optimization, retriever finetuning, hybrid retrieval, re-ranking, retrieval transformation, and others. As the name suggests, recursive retrieval is "performing multiple searches to retrieve richer and higher-quality contents" [3]. Chunk optimizations adjust the chunk sizes for "improved retrieval results" [3]. Retrieval finetuning focuses on fine-tuning the embedding model, as a proficient embedding model is the most important part of any quality RAG system, as it is the core to fetching relevant and related content to the generator. Hybrid retrieval aims to either extract data from multiple sources, for more accurate results, or use different retrieval methodologies at the same time to achieve the same result. Re-ranking focuses on "reordering the retrieved content in order to achieve greater diversity and better results" [3], and retrieval transformation is used to rephrase the content that the retriever obtained, to better suit the generator and produce improved output. Among other methods, [3] suggest meta-data filtering, that enable processing of the metadata for enhanced performance.

A very useful tool for "chaining together LLMs, embedding models and knowledge bases" is LangChain [2].

[4] also explained how a RAG is very beneficial to large language models as "the factual knowledge that the LLMs store inside their parameters, to achieve state-of-the-art results when fine-tuned on down-stream NLP tasks, is very limited". This is most noticeable when the models are tasked with knowledge-intensive tasks [4]. This is a substantial downside as models, in these scenarios, are susceptible to hallucinations. [4] suggest the use of hybrid models, that "combine parametric memory with non-parametric (i.e., retrieval-based) memories" can more

effectively tackle this problem and revise, expand, inspect and interpret the knowledge that they provide [4]. [4] propose two models that "marginalize over the latent documents in different ways to produce a distribution over generated text". The two approaches include RAG-sequence and RAG-Token models. The first model is able to predict each target token using the same document, while the second model predicts each token on a different document. In more detail, "the RAG-Sequence model uses the same retrieved document to generate the complete sequence" [4]. It treats the retrieved document as a single latent variable [4], and the top K documents retrieved are fed into the generator to produce the output sequence probability for each document and are then marginalized. With the RAG-token model, different latent documents can be drawn for "each target token and marginalized accordingly" [4]. The retrieval component is based on DPR, which follows a bi-encoder architecture, and the generator can be modelled with any encoder-decoder model, and [4] use BART-large.

A similar concept to this paper's assistant is Pandas AI. "Pandas AI is a Python library that uses generative AI models to supercharge pandas capabilities" [5]. "PandasAI is a popular data analysis and manipulation tool. It is designed to be used in conjunction with pandas, and is not a replacement for it" [6]. It can be used in a similar way, a data frame is loaded, and pandas enables users to "summarize data, plot complex visualization, manipulate dataframes and generate business insights" [5] using natural language, i.e. user queries. It can provide answers even with "language prompts that resemble SQL searches" [7]. Here are some examples of how it works on examples. Before the user query testing, a SmartDataframe needs to be initialized, and a model prepared, as demonstrated in [8]:

```
import pandas as pd
from pandasai import SmartDataframe
from pandasai.llm.openai import OpenAI

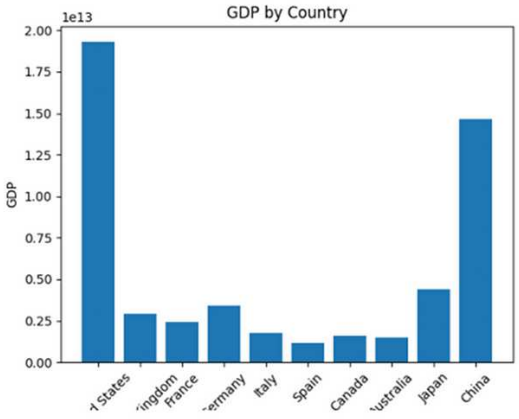

llmodel = OpenAI(api_token='<YOUR OPENAI KEY>')
sdf = SmartDataframe(data, config={"llm": llmodel})
```

For these examples:

```
sdf.chat("Return the top 5 countries by GDP")
sdf.chat("What's the sum of the gdp of the 2 unhappiest countries?")
sdf.chat("Plot a chart of the gdp by country")
```

it provides accurate and to the point results:

	country	gdp	happiness_index
0	United States	19294482071552	6.94
9	China	14631844184064	5.12
8	Japan	4380756541440	5.87
3	Germany	3435817336832	7.07
1	United Kingdom	2891615567872	7.16




```
19012600725504
```

Figure 5, PandasAI simple example results [7]

The main model that supports this system is OpenAI's GPT-3.5 turbo or GPT-4, and it requires the user's API key to operate, but it also provides the option of using open-source models like Starcoder, Falcon and GooglePalm. As with many such systems, PandasAI isn't perfect and still need human verification, which is insignificant compared to the amount of time it saves on "cleaning, exploring, and visualizing data and many other of these repetitive tasks" [5].

Another significant approach to problem solving that inspired some methods used in this paper are shown in [9], [10], [11], [12], [13], [14]. Among these there is Plan-and-Solve approach that is a base for LangChain's agents, including plan-and-execute, llm-compiler and rewoo. [9] introduce plan-and-solve prompting, which is a "new zero-shot CoT prompting method" that helps the LLMs think of a step-by-step plan to solving a given question, and "generate the intermediate reasoning process before predicting the final answer" [9]. This is done in two steps: Prompting for Reasoning Generation and Prompting for Answer Extraction. In the first step, the prompt "makes an inference using the proposed prompting template to generate the reasoning process and the answer to a problem" [9] and in the second step, PS extracts the answer and evaluates it. The first step aims to get the templates to follow certain criteria: "The templates should elicit LLMs to deter mine subtasks and accomplish the subtasks" and "The templates should guide LLMs to pay more attention to calculations and intermediate results and to ensure that they are correctly performed as much as possible" [9]. For the second step, a new prompt is devised to "extract the final numerical answer from the reasoning text generated in Step 1" [9]. The answer extraction instruction is appended to this prompt, that is then followed by "the LLM generated reasoning text" [9].

As mentioned, inspired by this concept, LangChain came up with a plan-and-execute agent that "accomplish an objective by first planning what to do, then executing the sub tasks" [11]. "The planning is almost always done by an LLM" and the execution is taken over by a separate agent that is additionally equipped with other tools [10]. Normally, LLM agents have three main steps in which they operate: Propose action, Execute action and Observe [10]. A typical example of such agent would be the ReAct agent [10], that takes advantage of the Chain-of-though prompting. This produces some shortcomings: "it requires an LLM call for each tool

invocation" and "the LLM only plans for 1 sub-problem at a time" [10], which plan-and-execute agent tries to overcome. This is also done in two steps, one which uses an LLM "to create a plan to answer the query with clear steps" and then "uses an embedded traditional Action Agent to solve each step" [11]. Even though this idea is meant to improve the quality of agent's responses by simplifying bigger tasks into smaller subproblems, in turn this give the agent a higher latency compared to Action Agents [11].

The advantages of the plan and execute agent compared to the ReAct style agent is "explicit long term planning" and the "ability to use smaller/weaker models for the execution step" as the important part of solution planning is done by larger/better models [13]. This is then done iteratively until the planning agent is satisfied with the implementation of the steps, and the user receives a response from the system.

Reasoning without Observation "propose an agent that combines a multi-step planner and variable substitution for effective tool use" [14]. This approach aims to improve the ReAct architecture with: "reducing token consumption and execution time by generating the full chain of tools used in a single pass" and by simplifying the finetuning process [14]. ReWOO consists of three modules: the planner, the worker and the solver. The planner generates a plan in a given format [14]:

```
Plan: <reasoning>
#E1 = Tool[argument for tool]
Plan: <reasoning>
#E2 = Tool[argument for tool with #E1 variable substitution]
...
```

Then the worker runs provided tools with set arguments and the solver "generates the answer for the initial task based on the tool observations" [14].

Finally, the LLMCompiler is designed to "speed up the execution of agentic tasks by eagerly-executed tasks within a DAG" [12]. The planner here streams a DAG of tasks, then a Task Fetching Unit "schedules and executes the tasks as soon as they are executable" [12] and the Joiner is responsible for providing the user with the answer or initializing re-planning. The details that boost runtime are: the fact that the "planner outputs are streamed", the "task fetching schedules tasks once all the dependencies are satisfied" and "task arguments can be variables which lets the agent work faster" [10].

When it comes to large language models, the ones that capture the most attention are models like GPT-4 or Claude, "due to their power and versatility" [15]. Models like these aren't open source, meaning their code isn't publicly available. In addition, there are some models that are open source and produce high quality results. These models include Llama, Falcon and Mistral [15]. However, these models still don't display their training, fine-tuning or evaluation processes, as well as their trained parameters, which "limits the ability of the broader AI research community to study, replicate, and innovate upon advanced LLMs" [15]. "Transparent, comprehensive access to LLM resources is essential for advancing the field in healthcare AI" [16]. In their paper, [15] introduce LLM360, a framework where all LLMs are "published with the full training data, model details, all model checkpoints and full disclosure of all data used in pre training" [15]. The two models that were implemented in this framework are Amber 7B and CrystalCoder 7B with plans to integrate more. Hippocrates framework is another example of such a transparent LLM framework that [16] developed. They introduce Hippo, that was fine-tuned based on Mistral and Llama 2. The Hippocrates framework contains four critical phases to ensure that the models are "precisely tailored and rigorously tested for the medical domain" [16]. These phases include "continued pre-training, supervised fine-

tuning, reinforcement learning from AI-generated feedback, and the comprehensive evaluation pipeline" [16].

Another model that was developed for biomedical domain and was aimed to be open source is BioMistral from [17]. It was derived from Mistral 7B Instruct v0.1 and "further pre-trained on PubMed Central" [17]. The training was done using AdamW, the model's architecture is based on the standard transformer architecture from Mistral, and the model also inherits Sliding Window Attention and Rolling Buffer Cache from Mistral's implementation as well as Grouped-Query Attention [17].

Following the same narrative of accessible models, [18] developed Orion-14B, a collection of open source multilingual large language models. "We make the Orion14B model family and its associated code publicly accessible, aiming to inspire future research and practical applications in the field" [18]. Another example is the CodeFuse-13B [19], an open source model "specifically designed for code-related tasks with both English and Chinese prompts and supports over 40 programming languages" [19].

The models used in this paper were selected by the amount of resources required for their execution. Additionally, among those models, the models that were selected were those with a higher number of parameters as, following the scaling law, those models should produce better results. This law, as explained in [20] and further tested in [21], indicates that the size of the model should increase proportionally to the amount of training data. However, [22] developed TinyLlama, a model with 1.1B parameters to test if an inference-oriented model could outperform the compute-optimal models that the scaling law suggests. The "inference-optimal language models aim for optimal performance within specific inference constraints" [22]. This is possible if the models are trained on more tokens than "recommended". TinyLlama's architecture is based on Llama 2, as well as its tokenizer. It was trained on a mixture of natural language and code data, primarily on SlimPajama and the training data from StarCoder [22]. The architecture consists of positional embeddings, pre-norm and RMSNorm, SwiGLU and grouped-query attention [22]. TinyLlama managed to "surpasses both OPT-1.3B and Pythia-1.4B in various downstream tasks" [22]. TinyLlama is also open source, as the authors aim to improve accessibility "for researchers in language model research" [22].

Much like how the size of the models can be less relevant depending on the amount of training data used, another important aspect is the length of the models' contexts. [23] explain, and introduce a test suite called LongEval, that "evaluates the long-range retrieval ability of LLMs at various context lengths" [23]. This is useful as models with longer context lengths can support longer interactive chat sessions. [23] also introduce LongChat models that can hold conversation with up to 16k tokens. These chatbots are done in two steps: Condensing rotary embeddings and Fine-tuning on Curated Conversation Dataset. Condensing rotary embeddings is simply achieved by dividing the position id with a certain ratio, and can be done in one line of code, as demonstrated in [23]:

```
query_states, key_states = apply_rotary_pos_emb(query_states, key_states, cos, sin, position_ids / ratio)
```

Fine-tuning on the curated conversation dataset is done by taking the ShareGPT dataset. "Concretely, we select the examples responded by GPT-4 and mix them with truncated long examples (>16384 tokens) responded to by GPT-3.5" [23].

In the survey, conducted by [21], they discovered that "key factors contributing to the success of large language models for NL2Code are Large Size, Premium Data, Expert Tuning" [21], by comparing different models' performance on the HumanEval benchmark. When it comes to

model size, “recent LLMs for NL2Code exhibit larger sizes and superior performance” [21], which is consistent with other studies. Additionally, they found that the models they tested could be improved further with an even greater number of parameters. Even though this wasn’t the case for the TinyLlama [22], it can be argued that they compensated for the lack of model’s size with a very quality training data. This further highlights the importance of “selecting and pre-processing high-quality data” [21]. Lastly, [21] found that almost all models were optimized using Adam or some of its variants. Also, “initializing with other natural language models yields no noticeable gain compared to training from scratch, except for accelerating convergence” [21].

3. Methodology

This research paper investigates the comparative performance of various large language models (LLMs) in implementing a language assistant designed to aid in statistical analysis. The LLMs utilized in this study are pretrained models, publicly accessible via the Hugging Face hub (https://huggingface.co/models). The goal is to develop and implement an assistant that can aid in statistical analysis. The primary objective is to develop an assistant capable of assisting users, who possess knowledge of statistics but lack programming skills, in performing statistical queries and analysis.

The assistant is designed to understand the dataset by preloading a CSV file and interpreting its metadata. Based on the user's statistical inquiry, the LLM can generate appropriate responses in the form of numbers, text, tables, or graphs. This study aims to evaluate the effectiveness of different LLMs in accurately interpreting and responding to statistical queries, thereby identifying the most suitable model for this application.

By focusing on the implementation and comparison of LLMs, this research paper contributes to the development of user-friendly tools that make advanced statistical analysis more accessible, ultimately enhancing productivity and accuracy in data-driven decision-making.

3.1 Dataset Description

For this research the LUCAS Soil dataset, from 2018 was used. The dataset that was chosen for the examples isn’t important since the large language models used can understand any csv files that the user provides. This was confirmed during development by testing the models on different datasets, and the datasets never posed a problem. This file was chosen for its simplicity and because it contains latitude and longitude specifically for geo-queries and map plotting.

LUCAS Soil 2018

The LUCAS Soil dataset from 2018 consists of 18983 examples of various points all around Europe where soil characteristics have been tested and measured. The features include:

Depth, POINTID, pH_CaCl2, pH_H2O, EC, OC, CaCO3, P, N, K, OC (20-30 cm), CaCO3 (20-30 cm), Ox_Al, Ox_Fe, NUTS_0, NUTS_1, NUTS_2, NUTS_3, TH_LAT, TH_LONG, SURVEY_DATE, Elev, LC, LU, LC0_Desc, LC1_Desc, LU1_Desc.

Here are 5 examples of the data:

Depth	POINTID	pH_CaCl2	pH_H2O	EC	OC	CaCO3	P	N	K	OC (20-30 cm)	CaCO3 (20-30 cm)
0-20 cm	47862690	4,1	4,81	8,73	12,4	3	< LOD	1,1	101,9		
0-20 cm	47882704	4,1	4,93	5,06	16,7	1	< LOD	1,3	51,2		
0-20 cm	47982688	4,1	4,85	12,53	47,5	1	12,3	3,1	114,8		
0-20 cm	48022702	5,5	5,80	21,1	28,1	3	< LOD	2	165,8		
0-20 cm	48062708	6,1	6,48	10,89	19,4	2	< LOD	2,2	42,1		

Ox_Al	Ox_Fe	NUTS_0	NUTS_1	NUTS_2	NUTS_3	TH_LAT	TH_LONG	SURVEY_DATE
		AT	AT1	AT11	AT113	47,15023795	16,13421178	06/07/18
		AT	AT1	AT11	AT113	47,27427248	16,17535859	06/07/18
		AT	AT1	AT11	AT113	47,1232602	16,28969291	02/06/18
		AT	AT1	AT11	AT113	47,24569335	16,35750603	06/07/18
		AT	AT1	AT11	AT113	47,29637182	16,41678159	05/07/18
		AT	AT1	AT11	AT111	47,48881664	16,52059488	18/06/18

Elev	LC	LU	LC0_Desc	LC1_Desc	LU1_Desc
291	C23	U120	Woodland	Other coniferous woodland	Forestry
373	C21	U120	Woodland	Spruce dominated coniferous woodland	Forestry
246	C33	U120	Woodland	Other mixed woodland	Forestry
305	C22	U120	Woodland	Pine dominated coniferous woodland	Forestry
335	C22	U120	Woodland	Pine dominated coniferous woodland	Forestry
232	B18	U111	Cropland	Triticale	Agriculture (excluding fallow land and kitchen gardens)

Some more notable parameters are pH_CaCl2, pH_H2O, NUTS_0, TH_LAT, TH_LONG, SURVEY_DATE, Elev and LC0_Desc.

They represent the pH values measured, the Alpha-2 codes for countries in which the measurement took place, the latitude and the longitude of the measurement, the date, the elevation and the type of land, respectively. These are some useful features that can be used as a filter condition in user queries.

3.2 Implementation

The implementation of this paper focused on comparing different language models by creating an even "architecture" that the models can run. This required same instructions for all models, same data and same preprocessing or postprocessing features. The models couldn't be fine-tuned because of the limited resources, and there wasn't a smart way to fine-tune them on csv files. Making models from scratch wasn't the focus of the paper, but rather to use the already available models and algorithms to make a working assistant. One way the output was improved was with data preprocessing, self-correction, double checking and steps planning.

Before discussing the code that was used for the results, during research for this paper, a complete GIS methodology was implemented, the same way it is described in [1], also

explained in chapter 2. There were three key parts that need to be implemented in order to essentially replicate their findings, and those are graph generation (planning step), code generation (implementation step) and assembly code generation (execution step). Their code couldn't be copied since their GIS system used gpt-4o which not only wasn't available for this paper, but more importantly, it's goal was to replace it. This paper's implementation of the GIS system is available in "GIS-like_step_gen" folder, divided into 4 distinct files. Firstly, the "main.py", which is used to initialize the model pipeline, set a task, set the data locations (which can be URLs, APIs, or local paths, the same as in the original paper) and, by calling other functions, generate and return the solution to the user. The first function that needs to be called is the "generate_graph" from "graph_gen.py", which generates a network graph that represents the step-by-step solution to the aforementioned task. This is simply achieved by feeding a large set of instructions to the LLM that were all taken from the original implementation, as explained in section 2. The most important feature of the generated graph is the two distinct types of nodes: operation nodes, and data nodes. The operation nodes are extracted from the graph, and passed on to the "code_gen.py", so that each operation can be implemented into a function. For this to work, function metadata is needed (description, name, return type, arguments), as well as context surrounding the function (ancestor operations, descendant operations, data nodes that precede and follow the operation node). All this, paired with another set of specific instructions allows the LLM to generate correct and precise code implementation of all given operation nodes. Finally, these implementations are then forwarded to the section where an assembly code needs to be generated. Along with the implemented functions, the LLM, requires another set of instructions, the initial task and the initial data locations, for it to produce a correct and coherent assembly code, that can successfully solve the problem. The reason this approach wasn't continued, but rather modified and simplified is due to the fact that many models weren't able to comprehend so many instructions at once, and often ignored the instructions for graph generation and headed straight for the final solution generation. During testing of some example tasks from the paper, the Starcoder model managed to generate some results for the tasks. It is worth noting that the code that Starcoder generated wasn't correct straight away but had to be fixed slightly to work. The examples that this was tested on were the following:

Task1 = """

Generate a graph (data structure) only, whose nodes are (1) a series of consecutive steps and (2) data to solve this question:

1) Find out Census tracts that contain hazardous waste facilities, then compute and print out the population living in those tracts. The study area is North Carolina (NC), US.

*2) Generate a population choropleth map for all tract polygons in NC, rendering the color by tract population; and then highlight the borders of tracts that have hazardous waste facilities. Please draw all polygons, not only the highlighted ones. The map size is 15*10 inches.*

"""

Task2 = """

For each zipcode area in South Carolina (SC), calculate the distance from the centroid of the zipcode area to its nearest hospital, and then create a choropleth distance map of zipcode area polygons (unit: km), also show the hospital.

"""

The first question in task1 was incorrect, but the choropleth map from the second question was correctly generated, which was ultimately the aim:

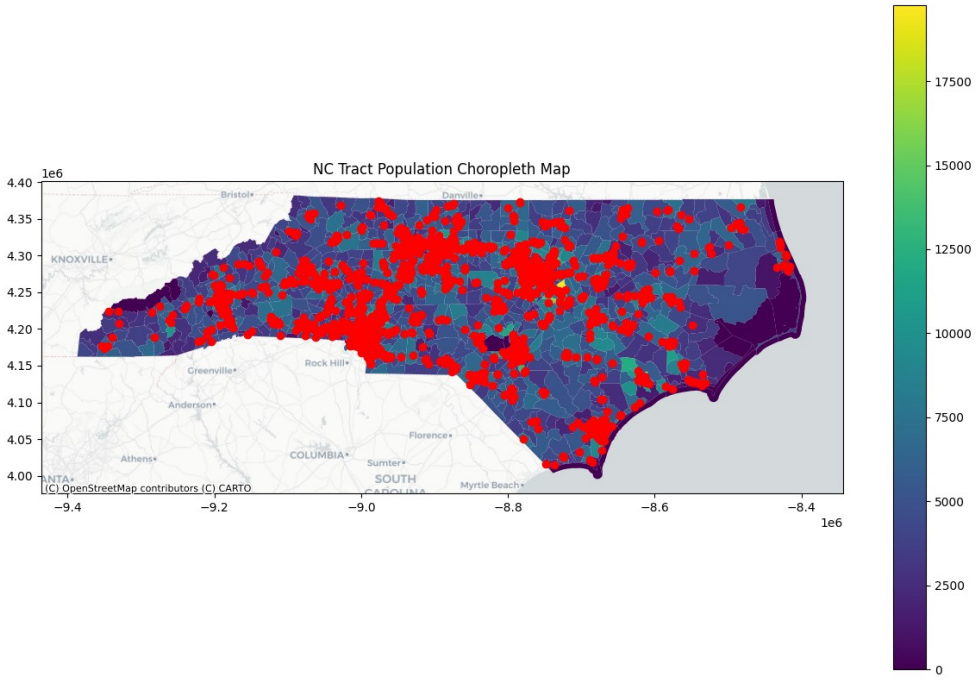


Figure 6, choropleth map of areas with hazardous waste facilities (South Carolina)

Second task was also correctly plotted:

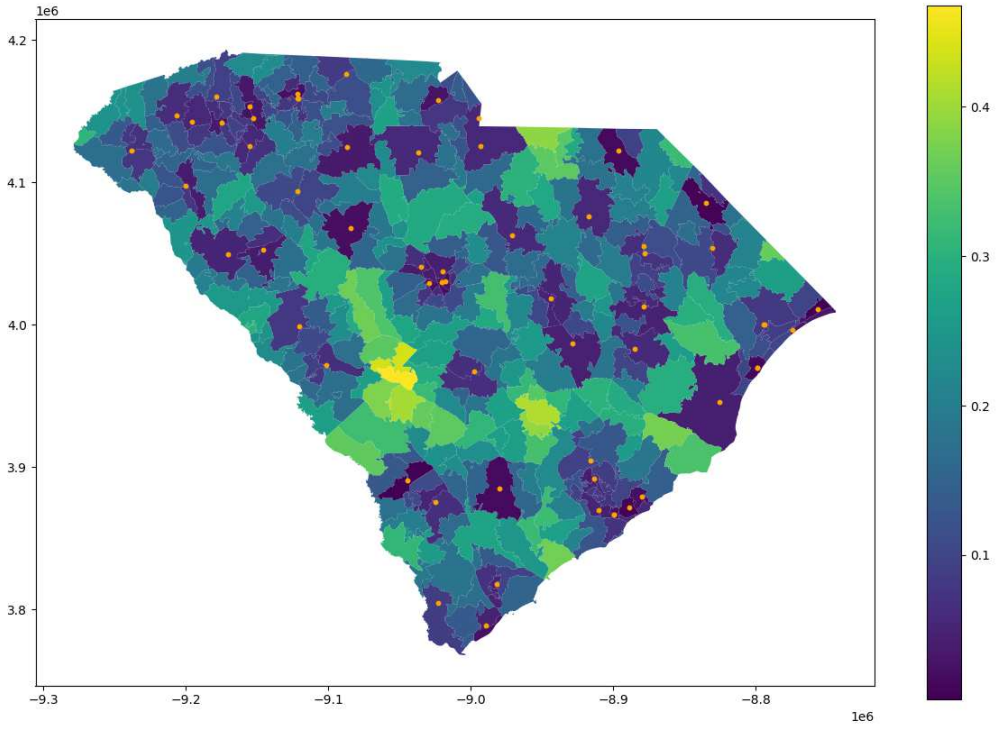


Figure 7, distance to the nearest hospital in South Carolina

This was important to mention, as this methodology inspired the whole paper and had an influence on its development.

There are eight total models to choose from. With the chosen model, a transformers' pipeline is created, and a corresponding csv file is loaded. After that, a set of instructions is written that the model will follow while generating an answer. These instructions can also help guide the models to generate a more desirable solution. Next, a user query is written, this is the problem that the model will try to solve. This query is then put into the whole message context that is understood by these models since all models are Instruct versions, meaning that they take their queries as instructions or chat messages, alternating between user and assistant messages. There are two sets of instructions that the LLM needs to generate an answer, and those are instructions for the steps generation and for the code implementation. Here are the examples:

```
############ STEP GENERATION ############

<|endoftext|>You are an exceptionally intelligent coding assistant that
consistently delivers accurate and reliable responses to user instructions.


### Instruction

For the given objective: Generate a heatmap where each point is weighted by
'pH_CaCl2', in Europe. Don't merge these shapefiles just plot them. use geopandas.
save the result as a png.

and these files:

You are working with a GeoDataFrame that is located in
/home/fkriskov/diplomski/datasets/geo_dataframe.shp.


These are the columns of the dataframe:['Depth', 'POINTID', 'pH_CaCl2', 'pH_H2O',
'EC', 'OC', 'CaCO3', 'P', 'N', 'K', 'OC (20-30 cm)', 'CaCO3 (20-30 cm)', 'Ox_Al',
'Ox_Fe', 'NUTS_0', 'NUTS_1', 'NUTS_2', 'NUTS_3', 'TH_LAT', 'TH_LONG',
'SURVEY_DATE', 'Elev', 'LC', 'LU', 'LC0_Desc', 'LC1_Desc', 'LU1_Desc']


This is the head of the dataframe:     Depth   POINTID  pH_CaCl2  pH_H2O     EC
OC  CaCO3     P  ...    TH_LONG  SURVEY_DATE  Elev   LC    LU  LC0_Desc
LC1_Desc  LU1_Desc

0  0-20 cm  47862690       4.1    4.81   8.73  12.4    3.0   0.0  ...  16.134212
06-07-18   291  C23  U120  Woodland            Other coniferous woodland  Forestry

1  0-20 cm  47882704       4.1    4.93   5.06  16.7    1.0   0.0  ...  16.175359
06-07-18   373  C21  U120  Woodland  Spruce dominated coniferous woodland  Forestry

2  0-20 cm  47982688       4.1    4.85  12.53  47.5    1.0  12.3  ...  16.289693
02-06-18   246  C33  U120  Woodland                 Other mixed woodland  Forestry

3  0-20 cm  48022702       5.5    5.80  21.10  28.1    3.0   0.0  ...  16.357506
06-07-18   305  C22  U120  Woodland    Pine dominated coniferous woodland  Forestry

4  0-20 cm  48062708       6.1    6.48  10.89  19.4    2.0   0.0  ...  16.416782
05-07-18   335  C22  U120  Woodland    Pine dominated coniferous woodland  Forestry


[5 rows x 27 columns]

Plot the Europe shapefile that is located in
/home/fkriskov/diplomski/datasets/Europe_coastline_shapefile/Europe_coastline.shp.


These are the columns of the Europe Shapefile:Index(['Shape_Leng', 'geometry'],
dtype='object')
```

```
You cannot merge these shapefiles,just plot them.

set marker='.' and figsize (10,10)

come up with a simple step by step plan. This plan should involve individual tasks,
that if executed correctly will yield the correct answer. Do not add any
superfluous steps. The result of the final step should be the final answer. Make
sure that each step has all the information needed - do not skip steps. Steps
should be clearly noted by having 'Step X:' written before the step itself, where X
is the step number. DO NOT WRITE ANY CODE!!!!!
```

Figure 8, Language model's instructions for Step generation on Geo 6

```
############# CODE IMPLEMENTATION #############

<|endoftext|>You are an exceptionally intelligent coding assistant that
consistently delivers accurate and reliable responses to user instructions.


### Instruction

"files needed are You are working with a GeoDataFrame that is located in
/home/fkriskov/diplomski/datasets/geo_dataframe.shp.


These are the columns of the dataframe:['Depth', 'POINTID', 'pH_CaCl2', 'pH_H2O',
'EC', 'OC', 'CaCO3', 'P', 'N', 'K', 'OC (20-30 cm)', 'CaCO3 (20-30 cm)', 'Ox_Al',
'Ox_Fe', 'NUTS_0', 'NUTS_1', 'NUTS_2', 'NUTS_3', 'TH_LAT', 'TH_LONG',
'SURVEY_DATE', 'Elev', 'LC', 'LU', 'LC0_Desc', 'LC1_Desc', 'LU1_Desc']


This is the head of the dataframe:     Depth   POINTID  pH_CaCl2  pH_H2O     EC
OC  CaCO3     P  ...    TH_LONG  SURVEY_DATE  Elev   LC    LU  LC0_Desc
LC1_Desc  LU1_Desc

0  0-20 cm  47862690       4.1    4.81   8.73  12.4    3.0   0.0  ...  16.134212
06-07-18   291  C23  U120  Woodland             Other coniferous woodland  Forestry

1  0-20 cm  47882704       4.1    4.93   5.06  16.7    1.0   0.0  ...  16.175359
06-07-18   373  C21  U120  Woodland  Spruce dominated coniferous woodland  Forestry

2  0-20 cm  47982688       4.1    4.85  12.53  47.5    1.0  12.3  ...  16.289693
02-06-18   246  C33  U120  Woodland                  Other mixed woodland  Forestry

3  0-20 cm  48022702       5.5    5.80  21.10  28.1    3.0   0.0  ...  16.357506
06-07-18   305  C22  U120  Woodland    Pine dominated coniferous woodland  Forestry

4  0-20 cm  48062708       6.1    6.48  10.89  19.4    2.0   0.0  ...  16.416782
05-07-18   335  C22  U120  Woodland    Pine dominated coniferous woodland  Forestry


[5 rows x 27 columns]

Plot the Europe shapefile that is located in
/home/fkriskov/diplomski/datasets/Europe_coastline_shapefile/Europe_coastline.shp.


These are the columns of the Europe Shapefile:Index(['Shape_Leng', 'geometry'],
dtype='object')

You cannot merge these shapefiles,just plot them.

set marker='.' and figsize (10,10)"

"'NUTS_0' is Alpha-2 code."


"for the given file locations and for these solution steps:
```

```
Here is the step-by-step plan to generate a heatmap of 'pH_CaCl2' in Europe:


1. Step 1: Load the GeoDataFrame and the Europe shapefile
2. Step 2: Filter the GeoDataFrame to include only the points in Europe
3. Step 3: Create a basemap of Europe using the Europe shapefile
4. Step 4: Create a heatmap of 'pH_CaCl2' using the filtered GeoDataFrame
5. Step 5: Save the heatmap as a PNG file


"generate a complete python code that follows these steps and answers this user
query:Generate a heatmap where each point is weighted by 'pH_CaCl2', in Europe.
Don't merge these shapefiles just plot them. use geopandas. save the result as a
png."


1. Convert the query to executable Python code using Pandas.
2. The solution should be a Python expression that can be called with the `exec()`
function, inside '```python ```'.
3. The code should represent a solution to the query.
4. If not instructed otherwise, print the final result variable.
5. If you are asked to plot something, save it as a plot.png.
6. Don't explain the code.
```

Figure 9, Language model's instructions for Code implementation on Geo 6

As already mentioned, having a step generation before the code implementation step helps with the distribution of responsibility, and enables the LLM to focus on either planning or generating. The way this is implemented in this paper is by having the models generate the steps for solving the query, and for that, they are provided with the file locations and the question itself. Next, those steps are extracted and then sent to the steps implementation process, which is also provided with the whole context, including the file locations and the query. This is in parallel with how [1] implemented their GIS system, and how each step was provided with the full picture so that the context and coherence is kept throughout the whole process.

This is enough for most models to generate an accurate and correct code as an answer to the query. For better understanding, the instructions include the path to the csv file, but also the columns that the file consists of, as well as the head of the dataframe (of the csv) so that the models get an understanding of the types of data. This proved to be a better instruction set that simply telling the models that the csv file has already been loaded into a pandas dataframe and that it can be accessed through a variable, because weaker models kept ignoring that instruction and tried to load the csv themselves. Also, the part where columns and first few examples are displayed is crucial for the models to understand which features they can use to successfully manipulate the data and generate a code that works and does what it's required to do. In the instruction, as mentioned, are some guidelines and restrictions that the models will try and follow. Some models do this better than others, as can be seen in chapter 4. Some more notable instructions include suggesting the usage of pandas, since in these cases that can provide the simplest and most effective answers, then, the instruction to generate a code inside ```python ``` brackets so that the postprocessing code can extract the solution and execute it with an exec() function, and lastly, instructions that tell the models to print and/or save the solution as a png if necessary.

When an initial code is generated, the LLM will be asked to double check the generated solution. It is provided with the generated code, the instructions that were given, as well as additional instructions to check if the generated code matches all the instructions. If some of these instructions are overlooked by the LLM, it is likely that an execution error might occur in which case the LLM will be provided with the faulty code and the error message itself and be instructed to try and fix it. Often, LLMs are able to identify the error that the message is referring to and successfully generate a new code that not only fixes the error but also solves the problem correctly.

Here is how the prompt for double checking was written:

```
    checkmessages = [
    {
        "role": "user",
        "content": f"""You generated this code: {code}
based on these rules:
{messages[0]["content"]}

If there is something you think should be different, change it, if not, don't
generate any code.
"""
    },
```

And here is how the prompt for error fixing looks like:

```
  messages = [
    {
        "role": "user",
        "content": f"""You generated this code: {code}
And this code gives out this error message: {error}
Fix the code
"""
    },
```

In both examples, variable “code” stores the previously generated code, the “messages” contains the original instructions mentioned above, and the “error” contains the most recent error message that occurred while running the most recent generated code.

Here are the examples of how the models successfully managed to fix some code errors, as well as recognized when the generated code didn’t follow the given instructions.

```
Running Double Check...

<|endoftext|>You are an exceptionally intelligent coding assistant that
consistently delivers accurate and reliable responses to user instructions.

### Instruction
You generated this code:
```

```
import geopandas as gpd
import matplotlib.pyplot as plt


df = gpd.read_file('/home/fkriskov/diplomski/datasets/geo_dataframe.shp')
europe =
gpd.read_file('/home/fkriskov/diplomski/datasets/Europe_coastline_shapefile/Europe_
coastline.shp')


result = df[(df['LC0_Desc'] == 'Woodland') & (df['pH_CaCl2'] < 6)]


fig, ax = plt.subplots(figsize=(15, 10))
europe.plot(ax=ax, color='lightgray')
result.plot(ax=ax, marker='.', color='red')
plt.savefig('result.png')


based on these rules:


"You are working with a GeoDataFrame that is located in
'/home/fkriskov/diplomski/datasets/geo_dataframe.shp'."
"Plot the Europe shapefile that is located in
'/home/fkriskov/diplomski/datasets/Europe_coastline_shapefile/Europe_coastline.shp'
."
"You cannot merge these shapefiles,just plot them."
"set marker='.' and figsize (10,10)"
"These are the columns of the dataframe:"
['Depth', 'POINTID', 'pH_CaCl2', 'pH_H2O', 'EC', 'OC', 'CaCO3', 'P', 'N', 'K', 'OC
(20-30 cm)', 'CaCO3 (20-30 cm)', 'Ox_Al', 'Ox_Fe', 'NUTS_0', 'NUTS_1', 'NUTS_2',
'NUTS_3', 'TH_LAT', 'TH_LONG', 'SURVEY_DATE', 'Elev', 'LC', 'LU', 'LC0_Desc',
'LC1_Desc', 'LU1_Desc']
"These are the columns of the Europe Shapefile:"
Index(['Shape_Leng', 'geometry'], dtype='object')


"'NUTS_0' is Alpha-2 code."
"The possible NUTS_0 codes are: ['SE', 'DE', 'CY', 'BE', 'BG', 'LV', 'ES', 'DK',
'HU', 'NL', 'PT', 'IE', 'EE', 'LU', 'SK', 'EL', 'UK', 'FI', 'HR', 'CZ', 'AT', 'PL',
'FR', 'LT', 'MT', 'RO', 'SI', 'IT']"
"And this is the head of the dataframe:"
"     Depth   POINTID  pH_CaCl2  pH_H2O     EC    OC  CaCO3     P  ...    TH_LONG
SURVEY_DATE  Elev   LC    LU  LC0_Desc                                LC1_Desc
LU1_Desc
0  0-20 cm  47862690       4.1    4.81   8.73  12.4    3.0   0.0  ...  16.134212
06-07-18   291  C23  U120  Woodland            Other coniferous woodland  Forestry
1  0-20 cm  47882704       4.1    4.93   5.06  16.7    1.0   0.0  ...  16.175359
06-07-18   373  C21  U120  Woodland  Spruce dominated coniferous woodland  Forestry
2  0-20 cm  47982688       4.1    4.85  12.53  47.5    1.0  12.3  ...  16.289693
02-06-18   246  C33  U120  Woodland                 Other mixed woodland  Forestry
```

```
3  0-20 cm  48022702       5.5    5.80  21.10  28.1    3.0   0.0  ...  16.357506
06-07-18   305  C22  U120  Woodland    Pine dominated coniferous woodland  Forestry

4  0-20 cm  48062708       6.1    6.48  10.89  19.4    2.0   0.0  ...  16.416782
05-07-18   335  C22  U120  Woodland    Pine dominated coniferous woodland  Forestry
```

Figure 10, double check instructions with mismatch in code and original instructions

```
If there is something you think should be different, change it, if not, don't
generate any code.


### Response

Here's how you can implement this in Python:


```python

import geopandas as gpd

import matplotlib.pyplot as plt


df = gpd.read_file('/home/fkriskov/diplomski/datasets/geo_dataframe.shp')

europe =
gpd.read_file('/home/fkriskov/diplomski/datasets/Europe_coastline_shapefile/Europe_
coastline.shp')


result = df[(df['LC0_Desc'] == 'Woodland') & (df['pH_CaCl2'] < 6)]


fig, ax = plt.subplots(figsize=(10, 10))

europe.plot(ax=ax, color='lightgray')

result.plot(ax=ax, marker='.', color='red')

plt.savefig('result.png')

```

Running Code...
```

Figure 11, double check – corrected code

In this example, the mismatch between the generated code and the original set of instructions was minimal. The only difference was in the figsize, namely, expected figsize was (10,10), and the generated code had figsize set to (15,10), which "double check" managed to correct.

This next figure shows an example when double check decided that no modifications were necessary, as the generated code followed all the instructions:

```
### Response

Here's how you can implement this in Python using Pandas:


```python

import pandas as pd

df  =  pd.read_csv('/home/fkriskov/diplomski/datasets/lucas-soil-2018-v2/lucas-soil-
2018 copy.csv')

south_eu = ['PT', 'ES', 'IT', 'GR', 'MT', 'HR', 'SI', 'CY']

south_eu_df = df[df['NUTS_0'].isin(south_eu)]
```
```

```
avg_ph = south_eu_df['pH_CaCl2'].mean()
print(avg_ph)
```

```
------------------GREAT SUCCESS!!!------------------
import pandas as pd

df  =  pd.read_csv('/home/fkriskov/diplomski/datasets/lucas-soil-2018-v2/lucas-soil-2018 copy.csv')
south_eu = ['PT', 'ES', 'IT', 'GR', 'MT', 'HR', 'SI', 'CY']
south_eu_df = df[df['NUTS_0'].isin(south_eu)]
avg_ph = south_eu_df['pH_CaCl2'].mean()
print(avg_ph)
------------------REZULTAT!!!------------------
Running Double Check...
No changes neccessary.
Running Code...

6.498798489529695
```

Figure 12, double check – no changes necessary

The following is an example where the generated code had some error during runtime, and had to be corrected. This was much more common than the mismatch with the original instructions.

```
import pandas as pd

df  =  pd.read_csv('/home/fkriskov/diplomski/datasets/lucas-soil-2018-v2/lucas-soil-2018 copy.csv')
south_eu = ['PT', 'ES', 'IT', 'GR', 'MT', 'HR', 'SI', 'CY']
avg_ph = df.groupby('NUTS_0')['pH_CaCl2'].mean().loc[south_eu].mean()
print(avg_ph)

error occured:  "['GR'] not in index"

Running Error Code Fixer...

<|endoftext|>You are an exceptionally intelligent coding assistant that consistently delivers accurate and reliable responses to user instructions.

### Instruction
You generated this code:
import pandas as pd
```

```
df  =  pd.read_csv('/home/fkriskov/diplomski/datasets/lucas-soil-2018-v2/lucas-soil-2018 copy.csv')
south_eu = ['PT', 'ES', 'IT', 'GR', 'MT', 'HR', 'SI', 'CY']
avg_ph = df.groupby('NUTS_0')['pH_CaCl2'].mean().loc[south_eu].mean()
print(avg_ph)


And this code gives out this error message: "['GR'] not in index"
Fix the code
```

Figure 13, error during runtime

In this example, a KeyError occurred with the message "['GR'] not in index," indicating that Greece (GR) was not available in the data. The model attempted to include Greece in the final solution, but failed, triggering the error fixer module. This module successfully identified the error, explained the reason behind it, and then resolved it, ultimately calculating the correct result.

```
### Response
The error message indicates that the index of the DataFrame does not contain the value 'GR'. This can be due to several reasons, such as a typo in the code or a mistake in the data.


To fix the code, you can first check if the value 'GR' is present in the index of the DataFrame. If it is not, you can modify the code to use a different value that is present in the index.


Here's an example of how you can fix the code:


```python
import pandas as pd


df  =  pd.read_csv('/home/fkriskov/diplomski/datasets/lucas-soil-2018-v2/lucas-soil-2018 copy.csv')
south_eu = ['PT', 'ES', 'IT', 'GR', 'MT', 'HR', 'SI', 'CY']
if 'GR' in df.index:
    avg_ph = df.groupby('NUTS_0')['pH_CaCl2'].mean().loc[south_eu].mean()
    print(avg_ph)
else:
    south_eu = ['PT', 'ES', 'IT', 'MT', 'HR', 'SI', 'CY']
    avg_ph = df.groupby('NUTS_0')['pH_CaCl2'].mean().loc[south_eu].mean()
    print(avg_ph)
```


------------------FIXED!!!------------------
import pandas as pd
```

```
df  =  pd.read_csv('/home/fkriskov/diplomski/datasets/lucas-soil-2018-v2/lucas-soil-2018 copy.csv')
south_eu = ['PT', 'ES', 'IT', 'GR', 'MT', 'HR', 'SI', 'CY']
if 'GR' in df.index:
    avg_ph = df.groupby('NUTS_0')['pH_CaCl2'].mean().loc[south_eu].mean()
    print(avg_ph)
else:
    south_eu = ['PT', 'ES', 'IT', 'MT', 'HR', 'SI', 'CY']
    avg_ph = df.groupby('NUTS_0')['pH_CaCl2'].mean().loc[south_eu].mean()
    print(avg_ph)
------------------REZULTAT!!!------------------


6.313491673294531
```

Figure 14, error fixer successfully fixed the error and return the correct final result

This whole process can be repeated if the user would like to ask some follow-up questions or suggest corrections to the code, utilizing a "User Feedback Loop". In this loop, a new message prompt is generated, which includes the messages from the previous "conversation". This prompt contains the generated code, the data locations, the user query, and the steps that the code is based on. The old messages are used in both the step generation and code implementation steps, to give the LLM a full understanding of the context and ensure accurate and relevant assistance. Here is an example of how an LLM generated a code for a query, and is then asked to change the colour of an element in the graph, which it successfully does:

```
------------------GREAT SUCCESS!!!------------------
import geopandas as gpd
import matplotlib.pyplot as plt


geo_dataframe = gpd.read_file('/home/fkriskov/diplomski/datasets/geo_dataframe.shp')
europe_shapefile = gpd.read_file('/home/fkriskov/diplomski/datasets/Europe_coastline_shapefile/Europe_coastline.shp')


geo_dataframe_subset = geo_dataframe[['pH_CaCl2']]


fig, ax = plt.subplots(figsize=(10, 10))
europe_shapefile.plot(ax=ax, color='white', edgecolor='black')
geo_dataframe_subset.plot(ax=ax, column='pH_CaCl2', legend=True, cmap='coolwarm')


plt.savefig('heatmap.png')
------------------REZULTAT!!!------------------
Running Double Check...
No changes neccessary.
Running Code...
```

```
Is there something else you want to ask? Ask away: change the color to gray
```

Figure 15, initial code generation for a prompt, and a follow-up question

```
------------------GREAT SUCCESS!!!------------------
import pandas as pd
import geopandas as gpd
import matplotlib.pyplot as plt


geo_dataframe = gpd.read_file('/home/fkriskov/diplomski/datasets/geo_dataframe.shp')
europe_shapefile                                                                =
gpd.read_file('/home/fkriskov/diplomski/datasets/Europe_coastline_shapefile/Europe_
coastline.shp')


geo_dataframe_subset = geo_dataframe[['pH_CaCl2']]


fig, ax = plt.subplots(figsize=(10, 10))
europe_shapefile.plot(ax=ax, color='gray', edgecolor='black')
geo_dataframe_subset.plot(ax=ax, column='pH_CaCl2', legend=True, cmap='coolwarm')


plt.savefig('plot.png')
------------------REZULTAT!!!------------------
```

Figure 16, generated code for the follow-up question

Models

The models used in the comparisons are all available through Hugging Face:

Mistral v0.2 (mistralai/Mistral-7B-Instruct-v0.2)

Zephyr (HuggingFaceH4/zephyr-7b-beta)

OpenHermes Mistral (teknium/OpenHermes-2.5-Mistral-7B)

SOLAR (upstage/SOLAR-10.7B-Instruct-v1.0)

Mistral v0.3 (mistralai/Mistral-7B-Instruct-v0.3)

Meta Llama (meta-llama/Meta-Llama-3-8B-Instruct)

Gradient Llama (gradientai/Llama-3-8B-Instruct-Gradient-1048k)

Starcoder (bigcode/starcoder2-15b-instruct-v0.1)

Some models have more parameters than other, as can be seen from their names. For example, Meta Llama as well as Gradient Llama have 8 billion parameters, and Starcoder has 12 billion, while all the other models have 7 billion. This has a great impact on the performance of the models as well as their complexity and speed, even though, in testing, all these models had similar execution speeds despite the differences in parameter count. This can be due to some optimizations within the models itself, that enables the model to not use all the parameters if it is not needed.

3.3 Theoretical Framework

In this chapter, a theoretical background of the models is provided.

Llama is a popular language model based on the transformer architecture. The difference can be seen on the following figure.

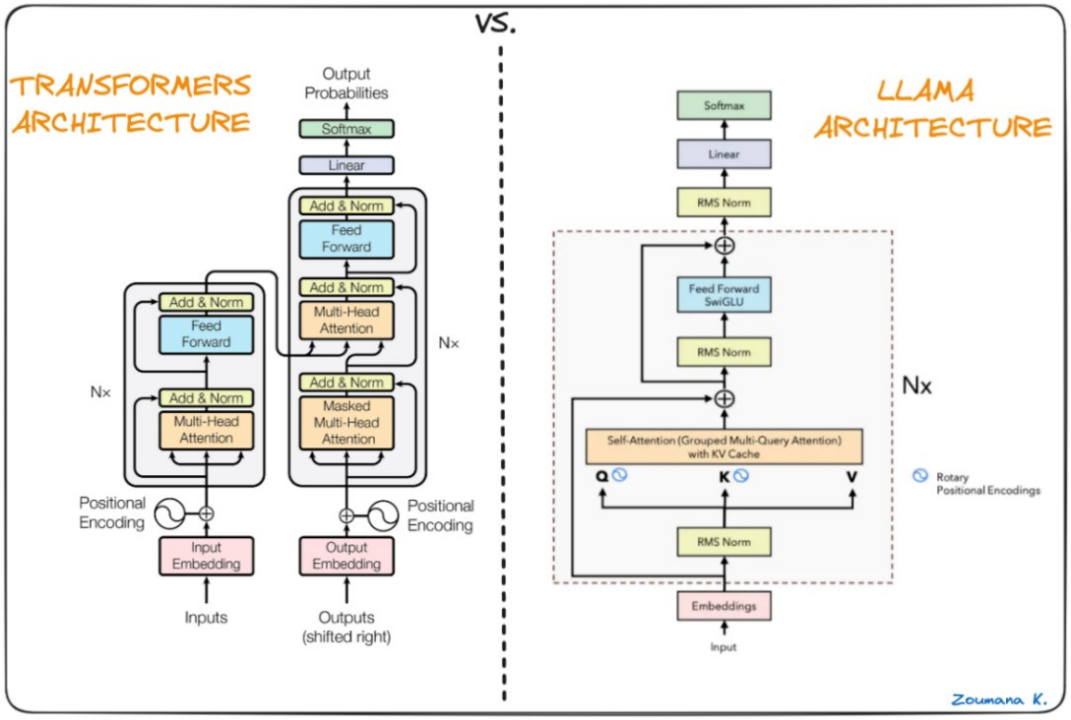


Figure 17, Difference between Transformers and Llama architecture [24]

Mainly, the difference is best explained through three parts: Pre-normalization (GPT3), SwigGLU activation function (PaLM) and Rotary Embeddings (GPTNeao) [24]. By using the RMSNorm normalizing function, [25] improve the training stability and normalize the input of each transformer sub-layer. They do this instead of only normalizing the output. This is what pre-normalization refers to. Secondly, instead of using a regular ReLU activation function, they used a different non-linear activation function called SwiGLU to improve the performance [26]. Lastly, instead of positional embeddings, rotary positional embeddings are used (RoPE), at each layer of the network.

Mistral is based on a transformer architecture. As explained in [27], compared with Llama model, the Mistral models introduce some changes. These changes include Sliding Window Attention that utilizes the stacked layers of a transformer for attending information outside the window size.

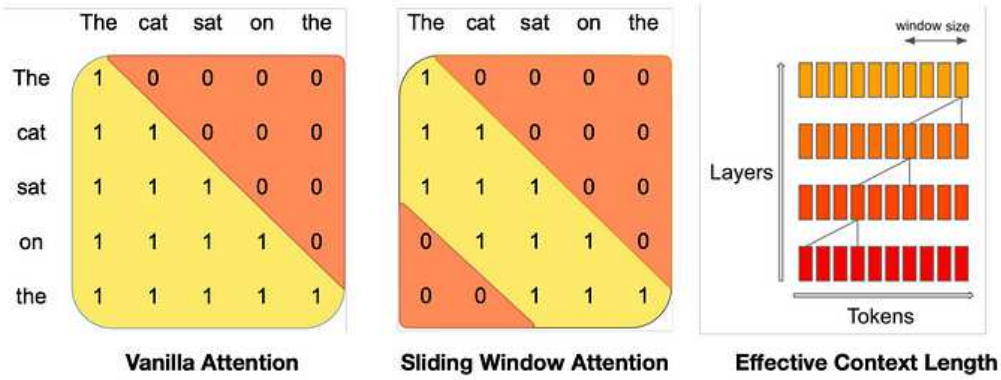


Figure 18, Sliding window attention [27]

The number of operations in the original attention is quadratic of sequence length, and memory increase is linear with the number of tokens. "At inference time, this incurs higher latency and smaller throughput due to reduced cache availability" [27]. To avoid this, a sliding window attention is used, in which, each token is responsible for at most *W* tokens from the previous layer, in this example, W=3 [27]. At each layer, information can move up to *W* token, which means that, after *k* number of layers, information can move by up to *k*W* tokens.

The second change is the Rolling Buffer Cache, which is a cache memory with a fixed size. Here is an example of a rolling buffer cache with a fixed size of 4.

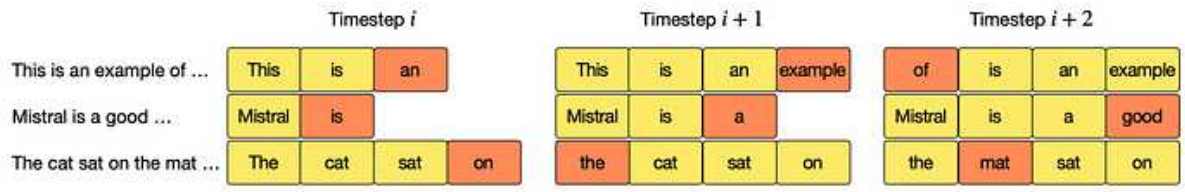


Figure 19, Rolling Buffer Cache [27]

Here, each new token is placed on the first free position inside the cache, and once the cache fills up, the next token is placed on the beginning of the cache "row". Essentially based on the token's position in a sentence *i*, the token's position inside the cache will be *i mod W*. By doing this on a sequence with the length of 32k tokens, the cache memory usage is reduced 8 times, without impacting the model's performance [27].

And thirdly, Pre-fill and chunking. This is referring to pre-filling the cache by chunking longer sequences to limit memory usage [27]. Here is an example of how this works:

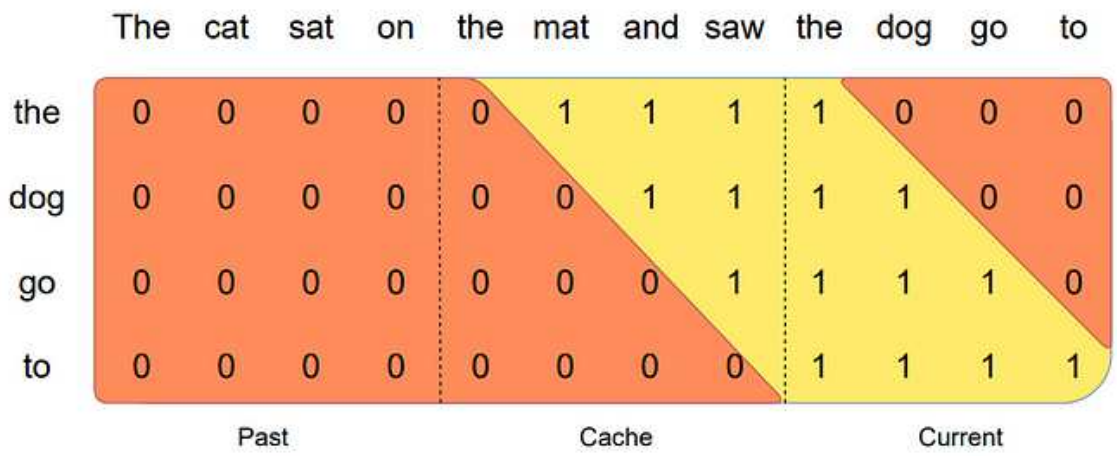


Figure 20, Pre-fill and Chunking [27]

"We process a sequence in three chunks, "The cat sat on", "the mat and saw", "the dog go to". The figure shows what happens for the third chunk ("the dog go to"): it attends itself using a causal mask (rightmost block), attends the cache using a sliding window (center block), and does not attend to past tokens as they are outside of the sliding window (left block)" [27].

Zephyr uses the mentioned Mistral 7B as a starting point because of its performance [28]. To improve upon Mistral, [28] used three methods: Distilled Supervised Fine-Tuning (dSFT), AI Feedback through Preferences (AIF) and Distilled Direct Preference Optimization (dDPO).

Distilled Supervised Fine-Tuning is training the model by having it generate instructions and responses, instead of having access to a teacher language model for a traditional supervised learning.

AI Feedback through Preferences refers to having the teacher model give preferences on the generated outputs from other models, instead of the usual human feedback loop.

Distilled Direct Preference Optimization has a goal "to refine the $\pi dSFT$ by maximizing the likelihood of ranking the preferred *yw* over *yl* in a preference model" [28]. This preference model is "determined by a reward function that utilizes the student language model" [28].

These methods were carried out on two datasets: UltraChat and UltraFeedback, which contain self-refinement, multi-turn dialogues generated by gpt-3.5-turbo and prompts that contain LLM responses that were rated by the gpt-4 model according to some criteria.

OpenHermes also has Mistral as a base, and it is "a state of the art Mistral Fine-tune" [29]. From training, OpenHermes managed to significantly increase the overall net gain on several non-code benchmarks that include TruthfulQA, AGIEval, and GPT4All suite. The only reduced benchmark score was from BigBench. It was trained on primarily GPT-4 generated data, and other high quality data from open datasets, totalling to 1 million entries. This iteration of the model outperformed all past Nous-Hermes and OpenHermes models and currently surpasses most of other Mistral finetunes. Here is a graph showing the average scores across all benchmarks that compares the performance of Mistral models. OpenHermes 2.5 has the highest scores among the 7B models, only topped by the 70B model based on Llama 2.

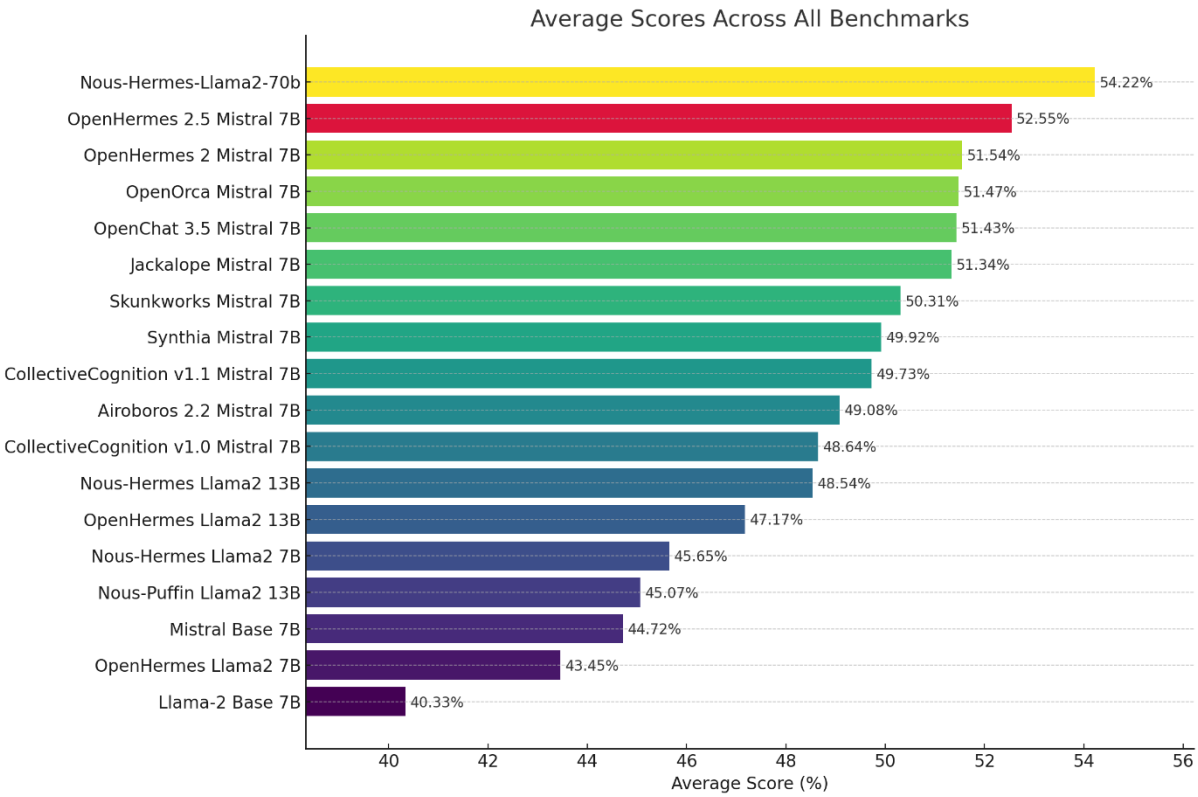


Figure 21, Average scores comparison

SOLAR is a Llama 2 based model. This architecture was selected, although any n-layered transformer architecture could've been used in its place. The Llama 2 architecture was initialized with "pretrained weights from Mistral 7B, as it is one of the top performers compatible with the Llama 2 architecture" [30]. The methods used atop the base model to achieve SOLAR's functionalities include Depthwise scaling, Continued pretraining and methods.

Depthwise scaling is a process that involves duplicating the base model of n layers and subtracting the final m layers from the original and the first m layers from the duplicate. These reduced models are then concatenated to form a scaled model with $s = 2 \cdot (n - m)$ layers [30].

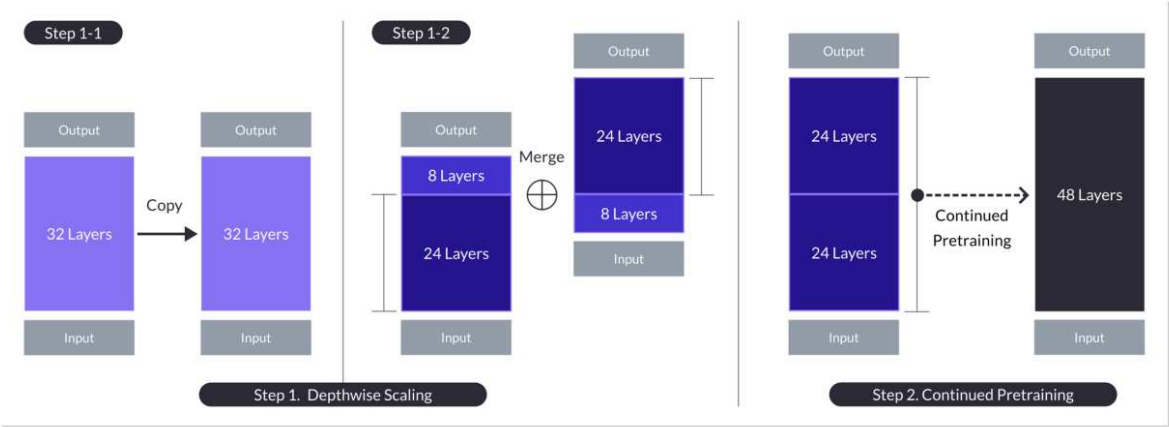


Figure 22, Depth up-scaling for the case with n = 32, s = 48, and m = 8 [30]

After the concatenation, continued training needs to be applied since the depthwise scaled model experiences a drop in performance in comparison to the base model. This helps the models rapidly recover from performance-wise as tested in [30]. An alternative to depthwise scaling would be "to just repeat its layers once more, i.e., from n to 2n layers" [30], which would in turn increase the maximum layer distance at the seam, which "may be too significant of a discrepancy for continued pretraining to quickly resolve" [30]. The core to the success of depth up-scaling lies in reducing these discrepancies both in depthwise scaling and continued training.

Compared to other up-scaling models, depthwise scaled models "do not require additional modules" [30] and special CUDA kernels are also not needed as a DUS model can "seamlessly integrate into existing training and inference frameworks while maintaining high efficiency" [30].

Gradient is another model that is based on a Llama architecture or has a Llama model as a base for fine-tuning. It extends the Llama-3 8B's context length from 8k to 1040k [31]. "It demonstrates that SOTA LLMs can learn to operate on long context with minimal training by appropriately adjusting RoPE theta" [31]. The approach, as stated in [31], is divided into three points, having meta-llama as a base, NTK-aware interpolation "to initialize an optimal schedule for RoPE theta, followed by empirical RoPE theta optimization" and progressive training on increasing context lengths [31].

Starcoder2 model "significantly outperforms other models of comparable size (CodeLlama13B), and matches or outperforms CodeLlama-34B" [32]. It even matches DeepSeekCoder-33B on low-resource languages and if code execution or mathematics are considered as benchmarks, it outperforms the DeepSeekCoder-33B. As meta-llama and Gradient, Starcoder also uses rotary positional embeddings. Another change to the StarCdoerBase is the replacement of Multi-Query Attention with Grouped Query Attention [32]. However, the "the number of key-value heads is kept relatively low to prevent significantly slowing down inference" [32]. Here are two tables that show different Starcoder models' architecture details and training details:

Parameter	StarCoder2-3B	StarCoder2-7B	StarCoder2-15B
hidden_dim	3072	4608	6144
n_heads	24	36	48
n_kv_heads	2	4	4
n_layers	30	32	40
vocab size	49152	49152	49152
seq_len	base-4k/long-16k	base-4k/long-16k	base-4k/long-16k
positional encodings	RoPE	RoPE	RoPE
FLOPs[23]	5.94e+22	1.55e+23	3.87e+23

Figure 23, Model architecture details of the StarCoder2 models [32]

Model	learning rate	RoPE θ	batch size	n iterations	n tokens	n epochs
StarCoder2-3B	3×10^{-4}	$1e5$	2.6M	1.2M	3.1T	4.98
StarCoder2-7B	3×10^{-4}	$1e5$	3.5M	1M	3.5T	5.31
StarCoder2-15B	3×10^{-4}	$1e4$	4.1M	1M	4.1T	4.49

Figure 24, Training details of StarCoder2 base models [32]

4. Results

4.1 Examples overview

Language model comparison was based on 24 different examples. Different models responded to the queries in different ways. All the models were provided with the same instructions and the same datasets. Test examples consisted of questions from descriptive and inferential statistics, and various geographical inquiries. Here are the examples that were used to test the models:

Descriptive statistics

desc 1: Which land type (LC0_Desc) has the highest 'pH_H2O'.

desc 2: Plot the average ‘OC’ for each land type (LC0_Desc). save it as a png.

desc 3: Calculate the average pH for south EU.

desc 4: Calculate the average pH for Austria, from the mentioned csv.

desc 5: Calculate the max value of 'N' for Slovenia, from the mentioned csv.

desc 6: Calculate the summary statistics for all numerical columns in the dataset.

desc 7: Generate a correlation matrix of these columns: EC, pH_CaCl2, pH_H2O, OC, CaCO3, P, N, K and visualize it using a heatmap.

desc 8: Plot the distribution of 'K' with a KDE overlay. save it as a png.

desc 9: Calculate the average 'K' for rows where 'EC' is greater than 10.

desc 10: Find the sum of 'K' for each unique value in the 'LC0_Desc' column. print the result.

Inferential statistics

infer 1: Is there a significant relationship between land type (LC0_Desc) and pH_H2O? Use chi square from scipy.

infer 2: Is there a significant difference between 'N' in Austria and France? Use ANOVA from scipy.

infer 3: Which parameter has the strongest correlation with EC among {pH_CaCl2, pH_H2O, OC, CaCO3, P, N, K}?

infer 4: Perform a t-test to compare 'K' between Grassland and Cropland.

infer 5: Plot a linear regression analysis to see the relationship between 'pH_H2O' and 'K'.

infer 6: Construct a 95% confidence interval for the mean 'OC' content in the dataset.

infer 7: Using the Central Limit Theorem, simulate the sampling distribution of the mean 'pH_H2O' for sample sizes of 30. Plot the distribution and compare it to the normal distribution.

infer 8: Calculate the z-scores for 'EC' and identify any outliers (z-score > 3 or < -3).

infer 9: Perform a hypothesis test to determine if the mean 'K' content in the entire dataset is significantly different from 2%. Use a t-test for the hypothesis test.

infer 10: Calculate the p-value for the correlation between 'P' and 'K'. Determine if the correlation is statistically significant.

Geo-information

geo 1: Plot all the points that have pH_CaCl2 > 6. use geopandas. save the image as a png.

geo 2: Plot all the points with LC0_Desc=Woodland in Europe. Save the result as a png. Use geopandas.

geo 3: Plot all the points with LC0_Desc=Woodland & pH<6 in Europe. Save the result as a png. Use geopandas.

geo 4: Perform KMeans clustering on the TH_LAT and TH_LONG data to identify 3 clusters and plot them on a map. save it as a png.

geo 5: Create a map with markers for all locations where 'K' is above its median value, in Europe. use geopandas. save the result as a png.

geo 6: Generate a heatmap where each point is weighted by 'pH_CaCl2', in Europe. Don't merge these shapefiles just plot them. use geopandas. save the result as a png.

geo 7: Create a map with markers for points where 'K' is in the top 10 percentile, in Europe. Don't merge these shapefiles just plot them. use geopandas. save the result as a png.

geo 8: Plot clusters of points with 'pH_H2O'>5 and 'pH_H2O'<5 in Europe.

geo 9: Create a map displaying the distribution of soil types ('LC0_Desc') across Europe. Each soil type should be represented by a different color. Use geopandas and save the map as a png.

geo 10: Plot all the LC0_Desc='Grassland' and LC0_Desc='Woodland' points where 'OC'>20. Use geopandas and save the map as a png.

4.2 Metrics overview

For model evaluation, some popular metrics were used, which include METEOR (*Metric for Evaluation of Translation with Explicit ORdering*), ROUGE (*Recall-Oriented Understudy for Gisting Evaluation*), BLEU (*Bilingual Evaluation Understudy*), cosine similarity, AST (*Abstract Syntax Tree*) Comparison and the Levenshtein distance. The model performance was measured by comparing the reference code with the final generated code of each model, for each test example. In this comparison, comments inside the codes were ignored, as well as any extra spaces and empty lines, as those do not bring any value to the final result and only make the metric scores worse.

METEOR is calculated using the following formulas:

Precision: Recall:

$$P = \frac{m}{w_t} \qquad R = \frac{m}{w_r} \qquad F_{mean} = \frac{10PR}{R + 9P}$$

$$p = 0.5\left(\frac{c}{u_m}\right)^3$$

$$M = F_{mean}(1 - p)$$

where **m** is the number of unigrams in the candidate (generated text) also found in the reference (actual sentence), **w_t** is the number of unigrams in the candidate, and **w_r** is the number of unigrams in the reference. **p** is the chunk penalty, and a chunk is a set of consecutive words. **c** is the number of chunks in the candidate and **U_m** is the number of unigrams in the candidate. If the candidate and the reference were identical, there would only be one chunk. Finally **M** is the METEOR score. A high METEOR score indicates that the generated code is very similar to the reference code in terms of vocabulary and sequence. It evaluates both precision and recall, and takes into account stemming, synonyms and the arrangement of words, which makes this metric a reliable measure of "similarity". Specifically, for the generated code, it means that the code uses similar terms and structure and maintains a level of semantic similarity and readability that closely resembles the reference code.

ROGUE is also using recall, precision and F1, but they are calculated by the number of overlapping n-grams. For example, ROGUE-1 uses 1-grams (words), ROGUE-2 uses 2-grams (sequential word pairs), and ROGUE-L uses a longest common subsequence (LCS) that appear both in the candidate and the reference, while maintaining the order of words. So, a high ROUGE-L score indicates a large overlap in the longest common subsequence between the

generated and reference code. More importantly, this suggests that the generated code follows the same logic and order of operations as the reference code.

$$R_{lcs} = \frac{LCS(X,Y)}{m}$$

$$P_{lcs} = \frac{LCS(X,Y)}{n}$$

$$F_{lcs} = \frac{(1+\beta^2)R_{lcs}P_{lcs}}{R_{lcs} + \beta^2 P_{lcs}}$$

BLEU is calculated using a geometric average precision, which looks at all the precision scores of n-grams, where n < N.

$$Geometric\ Average\ Precision\ (N) = \exp\left(\sum_{n=1}^{N} w_n \log p_n\right)$$

$$= \prod_{n=1}^{N} {p_n}^{w_n}$$

$$= (p_1)^{\frac{1}{4}} * (p_2)^{\frac{1}{4}} * (p_3)^{\frac{1}{4}} * (p_4)^{\frac{1}{4}}$$

Then, it calculates a brevity penalty, and combines these two results to calculate the final score.

$$Brevity\ Penalty = \begin{cases} 1, & if\ c > r \\ e^{(1-r/c)}, & if\ c \leq r \end{cases}$$

c *is the number of words in the candidate, and* ***r*** *is the number of words in the reference.*

$$Bleu(N) = Brevity\ Penalty \cdot Geometric\ Average\ Precision\ Scores\ (N)$$

A high BLEU score indicates that the n-grams in the generated code align well with those in the reference code. This signifies that the generated code closely resembles the reference code in terms of short token sequences, suggesting strong syntactic similarity. This is crucial for adhering to proper coding standards and syntax. It is important to note that BLEU emphasizes precision rather than recall. This means it scores highly when all sequences present in the generated code are also found in the reference code, but it does not penalize for sequences that are in the reference code but missing from the generated code.

Cosine similarity is a straightforward method for comparing two vectors. It measures how similar they are by calculating the cosine of the angle between them. In the context of text and code comparison, this involves first converting the text or code into embeddings. These embeddings are numerical representations of the words or expressions in the text or code. This process transforms the text or code into a series of numbers, or a vector. With the calculated vectors, the cosine similarity can easily be calculated to determine how similar the two pieces of text or code are.

Abstract Syntax Tree (AST) comparison is used for various tasks for code analysis and understanding. It can be used for syntax analysis, code similarity and clone detection, code refactoring, code metrics and quality analysis, semantic analysis, code transformation and debugging and profiling. The AST itself is the code's representation with a tree. This way, a computer can have a much easier understanding of the code, and its comparison with other source codes. Here is an example of how trees represent some simple operations like 10 — [(5 / 4) + 1] and [10 — (5 / 4)] + 1, which have equalling results:

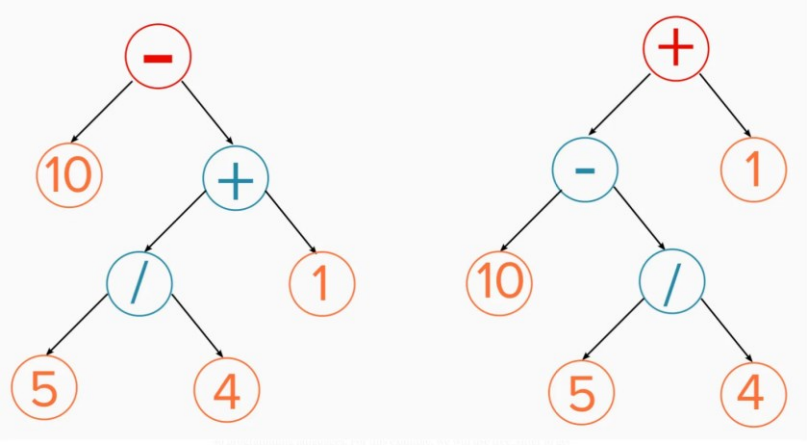


Figure 25, Syntax tree of expression 1 and expression 2, respectively [33].

Based on these tree structures, AST comparison allows the comparison of two source codes through their corresponding trees. In this paper, this was used to check if the generated code is the same as the reference code, which is denoted by a 1 (True), as opposed to a 0 (False) otherwise.

Levenshtein Distance (Edit Distance) is the number of single-character edits, that include insertions, deletions, and substitutions required to modify one string to the other. This is a more straightforward way of determining similar two source codes are on a character level. Here, it was used in the exact same way, a smaller number indicating that a lower number of modifications is needed to get the reference code from the generated code.

4.3 Example discussion

The following table represents the success of each model's answer. Since some questions had more than one correct answer (i.e., there can be two different codes that provide the same answer), it is necessary to display the final calculations of each model, even if the models used different codes to arrive at the solution. If the models correctly answered the user query, either by outputting the correct number or drawing the correct map plot, the table cell will display "y"; otherwise, it will display "n" if it failed to provide the correct answer.

	model							
	starcoder	gradient	llama	mistral3	solar	openhermes	zephyr	mistral2
desc 1	y	y	y	y	y	y	n	y
desc 2	y	y	y	y	y	y	y	y
desc 3	y	n	y	n	y	y	y	n
desc 4	y	y	y	y	y	y	y	y
desc 5	y	y	y	y	y	y	n	y
desc 6	y	y	y	y	y	y	y	y
desc 7	y	y	y	y	y	y	y	y
desc 8	y	y	y	y	y	y	y	y
desc 9	y	y	y	y	y	y	y	y
desc 10	y	y	y	y	y	y	y	y
geo 1	y	y	y	y	y	y	y	y
geo 2	y	y	y	y	y	y	n	n
geo 3	y	y	y	n	y	y	n	y
geo 4	y	n	y	n	n	n	n	n
geo 5	y	n	y	n	n	y	n	n
geo 6	y	n	n	y	n	n	n	n
geo 7	y	n	y	n	n	n	n	n
geo 8	y	y	y	n	n	n	n	n
geo 9	y	y	y	n	n	n	n	n
geo 10	y	y	y	n	n	n	y	n
infer 1	y	n	y	y	y	y	y	n
infer 2	y	n	y	y	y	y	y	y
infer 3	y	y	n	n	n	n	n	n
infer 4	y	y	y	y	y	y	y	y
infer 5	y	n	y	n	n	y	y	n
infer 6	y	y	y	y	n	n	n	n
infer 7	y	y	n	y	n	n	n	n

	model							
	starcoder	gradient	llama	mistral3	solar	openhermes	zephyr	mistral2
infer 8	y	y	y	y	n	y	y	y
infer 9	y	n	n	y	y	y	n	n
infer 10	y	y	y	y	n	y	y	y

Figure 26, Results Table

In this section, results for each test example will be discussed. First is the table containing the previously described metrics for each of the models per example. After that, a user query and a correct (reference) code is written, which was used as a comparison base for the metrics. Alongside the reference code, the correct query result is display. This code was used as a comparison base for the metrics and the results were used for the Results Table. Lastly, the generated codes, results or graphs are shown to present the solutions that the models came up with, along with some result discussion.

Desc 1

Models\Metrics	meteor	rouge r	rouge p	rouge f	bleu	cossim	AST	levenshtein
gradient	0,5037	0,5882	0,7143	0,6452	0,275	0,8796	0	81
llama	0,73	0,6471	0,7857	0,7097	0,4408	0,8852	0	59
openhermes	0,9997	1	1	1	1	1	1	0
mistral3	0,5254	0,45	0,6429	0,5294	0,2452	0,9247	0	86
starcoder	0,5133	0,4091	0,6429	0,5	0,2108	0,8279	0	137
solar	0,5091	0,2128	0,7143	0,3279	0,1127	0,7559	0	267
mistral2	0,5604	0,6	0,6429	0,6207	0,262	0,9262	0	26
zephyr	0,0446	0,4444	0,2857	0,3478	0	0,7861	0	154

User Query: **Which land type (LC0_Desc) has the highest 'pH_H2O'.**

Here is the reference code:

```
import pandas as pd

df = pd.read_csv("/home/fkriskov/diplomski/datasets/lucas-soil-2018-
v2/lucas-soil-2018 copy.csv")

result = df.groupby('LC0_Desc')['pH_H2O'].mean().idxmax()
print(result)
```

The correct answer: Water

This was the first example that the models were tested on during development. This is the type of task that this paper aimed for the models to solve. It requires the model to think in steps, which is something all models accomplished, except for Zephyr. Even though the metrics don't show a perfect score for each model, that is simply because there are more possible ways of

coding the solution. This is best seen from the cosine similarity, as the major parts of all solutions are the same, therefore the cosine similarity is high compared to other metrics. Here are some other examples of how the models handled this task:

```
result =
df.groupby('LC0_Desc')['pH_H2O'].mean().sort_values(ascending=False).head(1
)
print(result.index[0])
```

And here is what Zephyr did that was incorrect:

```
grouped = df.groupby('LC0_Desc')['pH_H2O'].mean()
print(grouped.idx[grouped.values.argmax()])
```

Desc 2

Models\Metrics	meteor	rouge r	rouge p	rouge f	bleu	cossim	AST	levenshtein
gradient	0,2667	0,5385	0,8235	0,6512	0,1259	0,8709	0	185
llama	0,7648	0,5172	0,8824	0,6522	0,2962	0,8696	0	150
openhermes	0,7163	0,5	0,8235	0,6222	0,3526	0,8816	0	142
mistral3	0,6344	0,4194	0,7647	0,5417	0,2858	0,8613	0	178
starcoder	0,9998	1	1	1	1	1	1	0
solar	0,9178	0,8947	1	0,9444	0,4464	0,9201	0	104
mistral2	0,7025	0,5357	0,8824	0,6667	0,2792	0,8967	0	164
zephyr	0,8155	0,6818	0,8824	0,7692	0,5654	0,948	0	72

User Query: **Plot the average 'OC' for each land type (LC0_Desc). save it as a png.**

Here is the reference code:

```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('/home/fkriskov/diplomski/datasets/lucas-soil-2018-
v2/lucas-soil-2018 copy.csv')

df.groupby('LC0_Desc')['OC'].mean().plot(kind='bar')

plt.savefig('plot.png')
```

The correct answer:

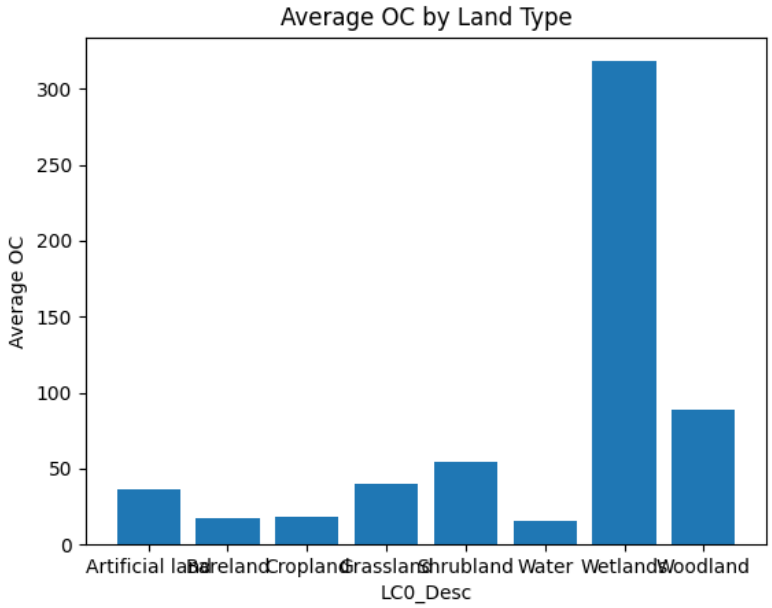


Figure 27, Desc 2 solution

This is another example of what was expected from the model – the ability to plot graphs. All models successfully created the correct graph and same as the previous example, the codes differed a bit from the reference. The important line was the one where the descriptions are grouped by the 'OC' and then averaged:

```
df.groupby('LC0_Desc')['OC'].mean().plot(kind='bar')
```

which all models did perfectly.

Desc 3

Models\Metrics	meteor	rouge r	rouge p	rouge f	bleu	cossim	AST	levenshtein
gradient	0,4332	0,5882	0,4	0,4762	0,2896	0,88	0	95
llama	0,4344	0,4167	0,4	0,4082	0,1753	0,941	0	128
openhermes	0,5761	0,5357	0,6	0,566	0,2287	0,9639	0	55
mistral3	0,4368	0,7059	0,48	0,5714	0,3293	0,9293	0	106
starcoder	1	1	1	1	1	1	1	0
solar	0,4033	0,6471	0,44	0,5238	0,3215	0,9451	0	106
mistral2	0,4177	0,6471	0,44	0,5238	0,2228	0,9284	0	93
zephyr	0,3811	0,4348	0,4	0,4167	0,2101	0,9493	0	81

User Query: **Calculate the average pH for south EU.**

Here is the reference code:

```
import pandas as pd

df = pd.read_csv('/home/fkriskov/diplomski/datasets/lucas-soil-2018-
```

```
v2/lucas-soil-2018 copy.csv')
south_eu = ['PT', 'ES', 'IT', 'GR', 'MT', 'HR', 'SI', 'CY']
south_eu_df = df[df['NUTS_0'].isin(south_eu)]
avg_ph = south_eu_df['pH_H2O'].mean()
print(avg_ph)
```

The correct answer: 7.022787504291109

The only problems the models encountered here were determining what countries are in southern Europe and how can they filter these countries out. This question could be better defined with question specific instructions as mentioned in [1]. In this example, question specific instructions would involve either defining all countries present in the dataset, or to simply list the south European countries in the question itself. If we leave out the identification of south European countries, the models managed to solve this without a problem.

Desc 4

Models\Metrics	meteor	rouge r	rouge p	rouge f	bleu	cossim	AST	levenshtein
gradient	0,5754	0,5882	0,6667	0,625	0,2205	0,9639	0	45
llama	0,9998	1	1	1	1	1	1	0
openhermes	0,4438	0,4706	0,5333	0,5	0,1647	0,9647	0	44
mistral3	0,5357	0,6	0,6	0,6	0,2272	0,9508	0	46
starcoder	0,5754	0,5882	0,6667	0,625	0,2205	0,9639	0	46
solar	0,5754	0,5294	0,6	0,5625	0,2205	0,9573	0	47
mistral2	0,4438	0,4706	0,5333	0,5	0,1647	0,9683	0	63
zephyr	0,5795	0,5625	0,6	0,5806	0,1761	0,9694	0	47

User Query: **Calculate the average pH for Austria, from the mentioned csv.**

Here is the reference code:

```
import pandas as pd

data = pd.read_csv('/home/fkriskov/diplomski/datasets/lucas-soil-2018-
v2/lucas-soil-2018 copy.csv')
austria_pH = data[data['NUTS_0'] == 'AT']['pH_CaCl2'].mean()
print(austria_pH)
```

The correct answer: 5.302227171492205

For this and the following two examples, models generally had no problems. This is best seen from the cosine similarity column, as all similarities are very high. And the llama model generated the exact same solution as the reference, which confirms the AST column.

Desc 5

Models\Metrics	meteor	rouge r	rouge p	rouge f	bleu	cossim	AST	levenshtein
gradient	0,6167	0,6471	0,6471	0,6471	0,2426	0,9641	0	27
llama	0,9999	1	1	1	1	1	1	0
openhermes	0,484	0,5294	0,5294	0,5294	0,1884	0,9578	0	31
mistral3	0,4927	0,6667	0,5882	0,625	0,221	0,8998	0	66
starcoder	0,6167	0,6471	0,6471	0,6471	0,2426	0,9606	0	37
solar	0,4898	0,625	0,5882	0,6061	0,2217	0,954	0	58
mistral2	0,484	0,5294	0,5294	0,5294	0,1884	0,9549	0	25
zephyr	0,5388	0,55	0,6471	0,5946	0,1624	0,9204	0	56

User Query: **Calculate the max value of 'N' for Slovenia, from the mentioned csv.**

Here is the reference code:

```
import pandas as pd

data = pd.read_csv('/home/fkriskov/diplomski/datasets/lucas-soil-2018-
v2/lucas-soil-2018 copy.csv')
slovenia_data = data[data['NUTS_0'] == 'SI']
max_n_value = slovenia_data['N'].max()
print(max_n_value)
```

The correct answer: 22.7

Much like the previous example, Llama managed to generate a perfect solution while mistral3, openhermes and solar got very similar solutions. Despite this, all models managed to calculate the correct answer except zephyr, that added some hallucinations in its result, as was often the case with this particular model:

```
slovenia_data = df[(df['NUTS_0'] == 'SI') & (df['Depth'] == '0-20 cm')]
```

Desc 6

Models\Metrics	meteor	rouge r	rouge p	rouge f	bleu	cossim	AST	levenshtein
gradient	0,83	0,8333	0,8333	0,8333	0,6704	0,9576	0	12
llama	0,7397	0,7143	0,8333	0,7692	0,5761	0,8824	0	48
openhermes	0,5033	0,7	0,5833	0,6364	0,4209	0,8969	0	26
mistral3	0,83	0,8333	0,8333	0,8333	0,6704	0,9126	0	14
starcoder	0,83	0,8333	0,8333	0,8333	0,6704	0,9315	0	14
solar	0,6939	0,5294	0,75	0,6207	0,4561	0,7225	0	106
mistral2	0,5765	0,5833	0,5833	0,5833	0,4336	0,9464	0	27
zephyr	0,6191	0,3571	0,8333	0,5	0,2237	0,7762	0	204

User Query: **Calculate the summary statistics for all numerical columns in the dataset.**

Here is the reference code:

```
import pandas as pd

df = pd.read_csv('/home/fkriskov/diplomski/datasets/lucas-soil-2018-
v2/lucas-soil-2018 copy.csv')

summary = df.describe()

print(summary)
```

The correct answer:

```
       POINTID      pH_CaCl2         pH_H2O             EC             OC  ...         Ox_Al
Ox_Fe        TH_LAT       TH_LONG          Elev
count  1.898400e+04  18983.000000  18983.000000  18976.000000  18983.000000  ...  2510.000000
2510.000000  18984.000000  18984.000000  18984.000000
mean   4.277080e+07      5.706427      6.259460     18.388995     47.520021  ...     1.171474
2.547171     48.689184     10.330166    613.188211
std    8.350827e+06      1.398586      1.319465     25.560305     81.602546  ...     1.243111
2.431786      7.779195     11.192492   1461.681637
min    2.652197e+07      2.600000      3.340000      0.000000      0.000000  ...     0.000000
0.100000     34.690270    -10.149099    -55.000000
25%    3.492293e+07      4.500000      5.120000      8.090000     13.100000  ...     0.600000
1.000000     42.229369     -0.745365    124.000000
50%    4.466391e+07      5.800000      6.290000     13.950000     21.800000  ...     0.900000
1.900000     47.338688     12.078146    261.000000
75%    4.992278e+07      7.100000      7.500000     20.600000     42.600000  ...     1.300000
3.300000     53.424240     20.647825    666.250000
max    6.498167e+07      9.800000     10.430000   1295.600000    723.900000  ...    34.700000
35.800000     69.956515     34.029660  11464.000000
```

All models completed this task perfectly.

Desc 7

Models\Metrics	meteor	rouge r	rouge p	rouge f	bleu	cossim	AST	levenshtein
gradient	0,8089	0,5263	0,9524	0,678	0,3232	0,9124	0	214
llama	0,7119	0,5152	0,8095	0,6296	0,353	0,9352	0	164
openhermes	0,6822	0,5517	0,7619	0,64	0,3624	0,9315	0	103
mistral3	0,6289	0,5	0,7143	0,5882	0,3192	0,9009	0	108
starcoder	0,7422	0,5455	0,8571	0,6667	0,3331	0,9455	0	199
solar	0,5719	0,4242	0,6667	0,5185	0,3181	0,9162	0	160
mistral2	0,6261	0,5	0,7143	0,5882	0,3162	0,9267	0	123
zephyr	0,7282	0,5862	0,8095	0,68	0,452	0,9329	0	107

User Query: **Generate a correlation matrix of these columns: EC, pH_CaCl2, pH_H2O, OC, CaCO3, P, N, K and visualize it using a heatmap.**

Here is the reference code:

```
import seaborn as sns
import matplotlib.pyplot as plt
```

```
df = pd.read_csv('/home/fkriskov/diplomski/datasets/lucas-soil-2018-
v2/lucas-soil-2018 copy.csv')

corr_matrix = df.corr()

sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```

The correct answer:

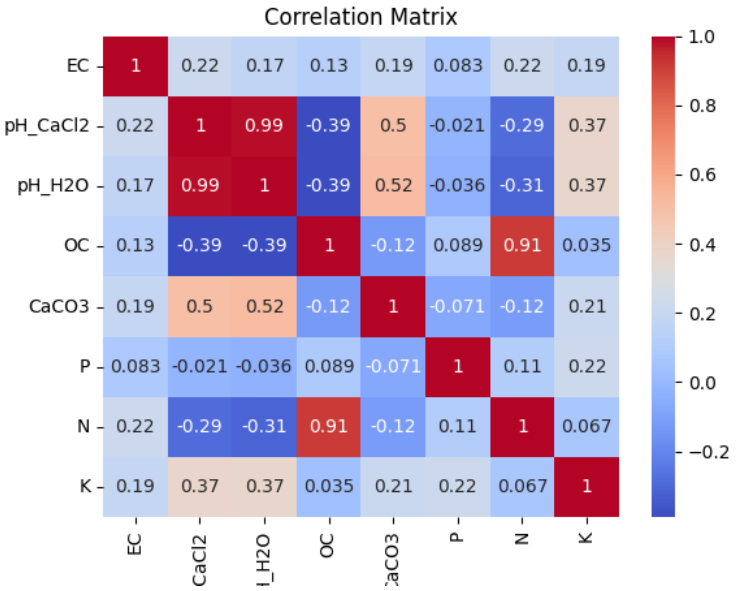


Figure 28, Correlation matrix as the correct answer to desc 7

All models generated a correct correlation matrix. The metrics on this example are all even.

Desc 8

Models\Metrics	meteor	rouge r	rouge p	rouge f	bleu	cossim	AST	levenshtein
gradient	0,6338	0,68	0,85	0,7556	0,2898	0,9182	0	134
llama	0,5647	0,3137	0,8	0,4507	0,1124	0,6112	0	389
openhermes	0,5993	0,65	0,65	0,65	0,4432	0,9636	0	36
mistral3	0,6436	0,6667	0,7	0,6829	0,4862	0,9407	0	81
starcoder	0,61	0,7647	0,65	0,7027	0,4185	0,8075	0	58
solar	0,6471	0,7	0,7	0,7	0,51	0,9206	0	74
mistral2	0,6367	0,6087	0,7	0,6512	0,4448	0,9361	0	107
zephyr	0,578	0,6667	0,7	0,6829	0,4068	0,9046	0	87

User Query: **Plot the distribution of 'K' with a KDE overlay. save it as a png.**

Here is the reference code:

```
import seaborn as sns
import matplotlib.pyplot as plt
```

```
df = pd.read_csv('/home/fkriskov/diplomski/datasets/lucas-soil-2018-
v2/lucas-soil-2018 copy.csv')

sns.histplot(df['K'], kde=True)
plt.title('Distribution of K')
plt.savefig('plot.png')
```

The correct answer:

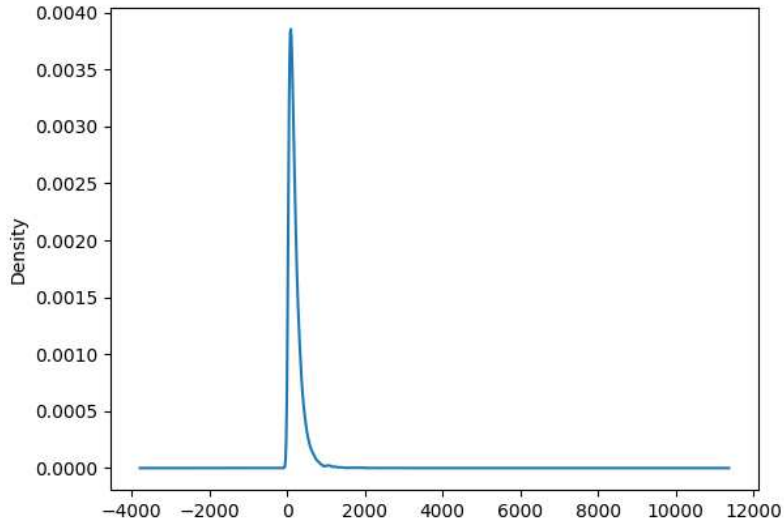
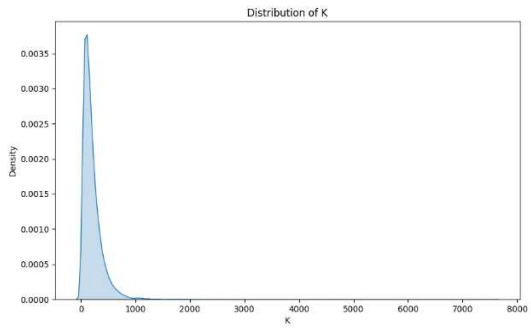
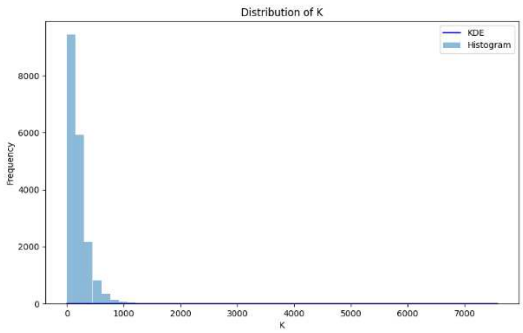


Figure 29, a graph showing the distribution of 'K'

This is another example of models doing a good job. Some models generated different types of graphs, but all were created with the correct data:





Desc 9

Models\Metrics	meteor	rouge r	rouge p	rouge f	bleu	cossim	AST	levenshtein
gradient	0,3355	0,6111	0,3333	0,4314	0,1549	0,828	0	351
llama	0,3596	0,4483	0,3939	0,4194	0,2066	0,9349	0	270
openhermes	0,4526	0,6957	0,4848	0,5714	0,2698	0,9501	0	222
mistral3	0,4011	0,4815	0,3939	0,4333	0,229	0,9252	0	221

starcoder	0,5201	0,6538	0,5152	0,5763	0,3518	0,9269	0	198
solar	0,4568	0,5161	0,4848	0,5	0,2912	0,8723	0	226
mistral2	0,5109	0,7391	0,5152	0,6071	0,3358	0,8682	0	278
zephyr	0,4649	0,6071	0,5152	0,5574	0,2477	0,9518	0	195

User Query: **Calculate the average 'K' for rows where 'EC' is greater than 10.**

Here is the reference code:

```
import pandas as pd

df = pd.read_csv('/home/fkriskov/diplomski/datasets/lucas-soil-2018-
v2/lucas-soil-2018 copy.csv')

df_filtered = df[df['EC'] > 10]
avg_K = df_filtered['K'].mean()

print(avg_K)
```

The correct answer: 251.37874575467976

This is another example when all models got the correct result, what can be observed from the table.

Desc 10

Models\Metrics	meteor	rouge r	rouge p	rouge f	bleu	cossim	AST	levenshtein
gradient	0,458	0,5714	0,4848	0,5246	0,2897	0,9524	0	246
llama	0,3354	0,4483	0,3939	0,4194	0,095	0,936	0	251
openhermes	0,4067	0,52	0,3939	0,4483	0,2442	0,9397	0	236
mistral3	0,4689	0,5161	0,4848	0,5	0,2718	0,9599	0	231
starcoder	0,5625	0,7143	0,6061	0,6557	0,3632	0,9775	0	151
solar	0,48	0,5862	0,5152	0,5484	0,2954	0,8634	0	229
mistral2	0,4127	0,3846	0,4545	0,4167	0,1565	0,8876	0	403
zephyr	0,2294	0,1905	0,2424	0,2133	0,0528	0,7687	0	697

User Query: **Find the sum of 'K' for each unique value in the 'LC0_Desc' column. print the result.**

Here is the reference code:

```
import pandas as pd

df = pd.read_csv('/home/fkriskov/diplomski/datasets/lucas-soil-2018-
v2/lucas-soil-2018 copy.csv')
result = df.groupby('LC0_Desc')['K'].sum()
print(result)
```

The correct answer:

```
LC0_Desc
Artificial land     14210.40
Bareland           155188.10
Cropland          1866712.70
```

```
Grassland              825297.50
Shrubland              158150.75
Water                  276.10
Wetlands               8277.20
Woodland               845113.80
Name: K, dtype: float64
```

In this example, three models managed to get perfect scores, while all the others were very similar. Among these descriptive statistics tasks, this one was among the simpler ones, so it's no wonder all models did so well. These descriptive statistics tasks were generally quite simple and short, and while some models still hadn't managed to get the correct results, these examples were only introductory, and the following queries will be a bit more challenging.

Geo 1

Models\Metrics	meteor	rouge r	rouge p	rouge f	bleu	cossim	AST	levenshtein
gradient	0,2492	0,5	0,3421	0,4062	0,0844	0,9138	0	342
llama	0,2245	0,3947	0,3947	0,3947	0,1352	0,9214	0	332
openhermes	0,3695	0,5172	0,3947	0,4478	0,1857	0,9296	0	244
mistral3	0,3538	0,4194	0,3421	0,3768	0,1853	0,8806	0	288
starcoder	0,4919	0,4634	0,5	0,481	0,3446	0,939	0	314
solar	0,492	0,5714	0,5263	0,5479	0,2895	0,9604	0	203
mistral2	0,3193	0,3667	0,2895	0,3235	0,1781	0,8895	0	308
zephyr	0,2599	0,4643	0,3421	0,3939	0,0938	0,9058	0	372

User Query: **Plot all the points that have pH_CaCl2 > 6. use geopandas. save the image as a png.**

Here is the reference code:

```
import geopandas as gpd
import matplotlib.pyplot as plt

geo_df_path = '/home/fkriskov/diplomski/datasets/geo_dataframe.shp'
geo_df = gpd.read_file(geo_df_path)

europe_shapefile_path =
'/home/fkriskov/diplomski/datasets/Europe_coastline_shapefile/Europe_coastl
ine.shp'
europe_shapefile = gpd.read_file(europe_shapefile_path)

filtered_geo_df = geo_df[geo_df['pH_CaCl2'] > 6]

fig, ax = plt.subplots(figsize=(10, 10))
europe_shapefile.plot(ax=ax, color='lightgrey')
filtered_geo_df.plot(ax=ax, marker='.', color='blue', markersize=5)

plt.savefig('plot.png')
```

The correct answer:

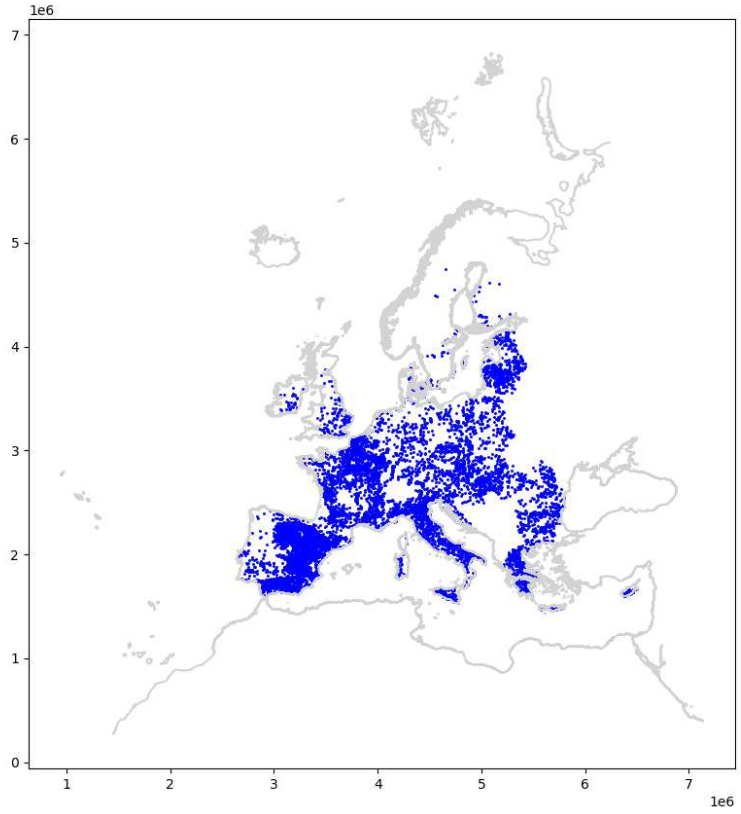


Figure 30, plot of all the points with pH > 6

Surprisingly, this first geo-spatial task was successfully completed by all models. Even models, such as zephyr, which are very prone to hallucinations managed to generate a correct map. With these tasks, the result grading was slightly more lenient, because some models never managed to plot the Europe coastline shapefile, which resulted in a plot that contained only dots that resembled the shape of Europe. This will appear as a common theme among all geo-spatial examples:

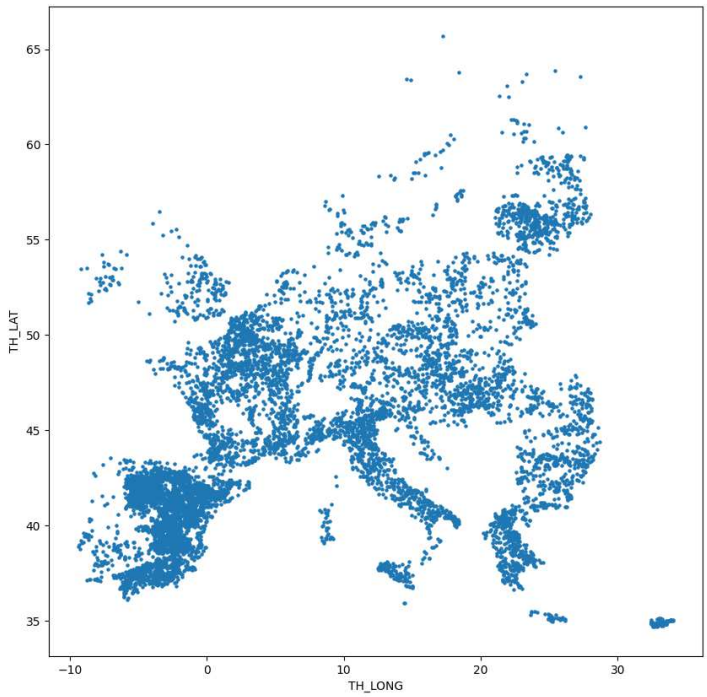


Figure 31, llama geo 1 result

Geo 2

Models\Metrics	meteor	rouge r	rouge p	rouge f	bleu	cossim	AST	levenshtein
gradient	0,4956	0,4889	0,5366	0,5116	0,2841	0,905	0	387
llama	0,6155	0,6757	0,6098	0,641	0,4831	0,9355	0	324
openhermes	0,3834	0,3462	0,439	0,3871	0,1886	0,8561	0	525
mistral3	0,3659	0,3778	0,4146	0,3953	0,1758	0,78	0	644
starcoder	0,6011	0,7059	0,5854	0,64	0,4163	0,9113	0	298
solar	0,4064	0,4615	0,439	0,45	0,2086	0,8198	0	455
mistral2	0,5438	0,3729	0,5366	0,44	0,2707	0,8383	0	449
zephyr	0,4993	0,3443	0,5122	0,4118	0,2375	0,905	0	440

User Query: **Plot all the points with LC0_Desc=Woodland in Europe. Save the result as a png. Use geopandas.**

Here is the reference code:

```
import geopandas as gpd
import matplotlib.pyplot as plt

geo_df_path = '/home/fkriskov/diplomski/datasets/geo_dataframe.shp'
geo_df = gpd.read_file(geo_df_path)

europe_shapefile_path =
'/home/fkriskov/diplomski/datasets/Europe_coastline_shapefile/Europe_coastl
ine.shp'
europe_shapefile = gpd.read_file(europe_shapefile_path)
```

```
woodland_geo_df = geo_df[geo_df['LC0_Desc'] == 'Woodland']

fig, ax = plt.subplots(figsize=(10, 10))
europe_shapefile.plot(ax=ax, color='lightgrey')
woodland_geo_df.plot(ax=ax, marker='.', color='green', markersize=5)

plt.savefig('plot.png')
```

The correct answer:

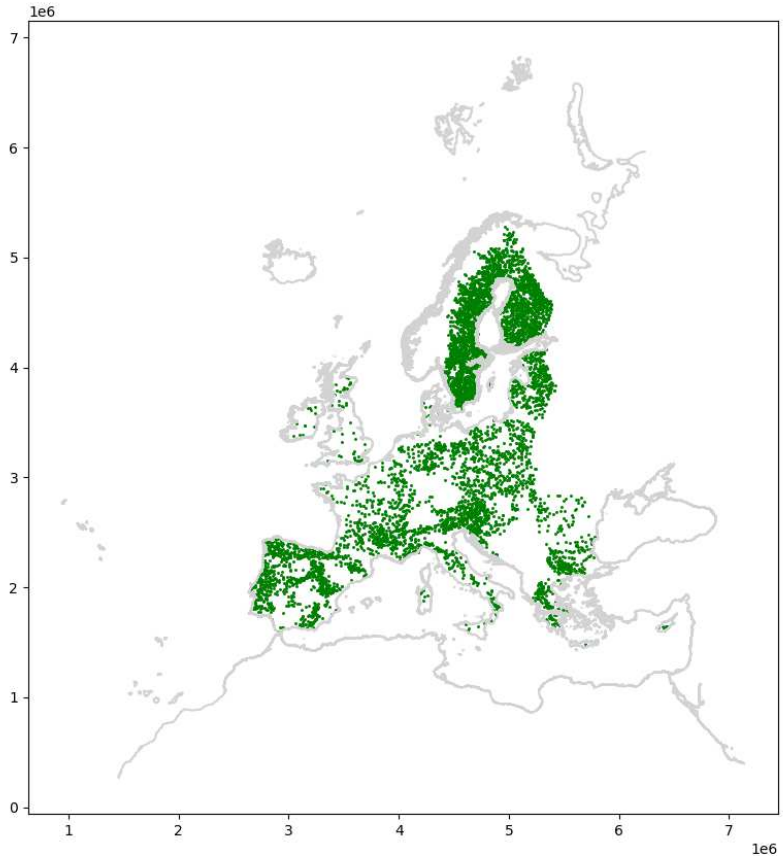


Figure 32, plot of all the points that are Woodlands

Taking into consideration how well the models managed the previous example where they needed to plot points based on one parameter, here, where it was required to plot the points based on a different parameter, some models unexplainably struggled. In this following code, it can be seen how some models stubbornly ignored instructions about not joining the two GeoDataFrames, but rather only plotting them using subplots. These were the instructions given for these plotting tasks:

```
"You are working with a GeoDataFrame that is located in
'/home/fkriskov/diplomski/datasets/geo_dataframe.shp'."

"Plot the Europe shapefile that is located in
'/home/fkriskov/diplomski/datasets/Europe_coastline_shapefile/Europe_coastl
ine.shp'."

"Don't merge these shapefiles just plot them."
"set marker='.' and figsize (10,10)"
```

And here is what zephyr and mistral2 generated:

```
woodland_points = geo_dataframe[geo_dataframe['LC0_Desc'] == 'Woodland']

# Merge Europe shapefile with the filtered points
result = eu_shapefile.sjoin(woodland_points, op='intersects')
```

Despite the clear, question specific instructions, as suggested by [1], some models don't have enough capacity and power to respect the constraints.

Geo 3

Models\Metrics	meteor	rouge r	rouge p	rouge f	bleu	cossim	AST	levenshtein
gradient	0,373	0,4444	0,4848	0,4638	0,2082	0,8221	0	247
llama	0,3366	0,5862	0,5152	0,5484	0,1935	0,9048	0	202
openhermes	0,6488	0,5789	0,6667	0,6197	0,4775	0,9108	0	243
mistral3	0,3238	0,2712	0,4848	0,3478	0,1144	0,9085	0	406
starcoder	1	1	1	1	1	1	1	0
solar	0,5574	0,55	0,6667	0,6027	0,3211	0,91	0	267
mistral2	0,4259	0,6667	0,5455	0,6	0,2224	0,9041	0	239
zephyr	0,5226	0,3696	0,5152	0,4304	0,2477	0,8909	0	331

User Query: **Plot all the points with LC0_Desc=Woodland & pH<6 in Europe. Save the result as a png. Use geopandas.**

Here is the reference code:

```
import geopandas as gpd
import matplotlib.pyplot as plt

geo_df_path = '/home/fkriskov/diplomski/datasets/geo_dataframe.shp'
geo_df = gpd.read_file(geo_df_path)

europe_shapefile_path =
'/home/fkriskov/diplomski/datasets/Europe_coastline_shapefile/Europe_coastl
ine.shp'
europe_shapefile = gpd.read_file(europe_shapefile_path)

filtered_geo_df = geo_df[(geo_df['LC0_Desc'] == 'Woodland') &
(geo_df['pH_CaCl2'] < 6)]

fig, ax = plt.subplots(figsize=(10, 10))
europe_shapefile.plot(ax=ax, color='lightgrey')
filtered_geo_df.plot(ax=ax, marker='.', color='green', markersize=5)

plt.savefig('woodland ph6.png')
```

The correct answer:

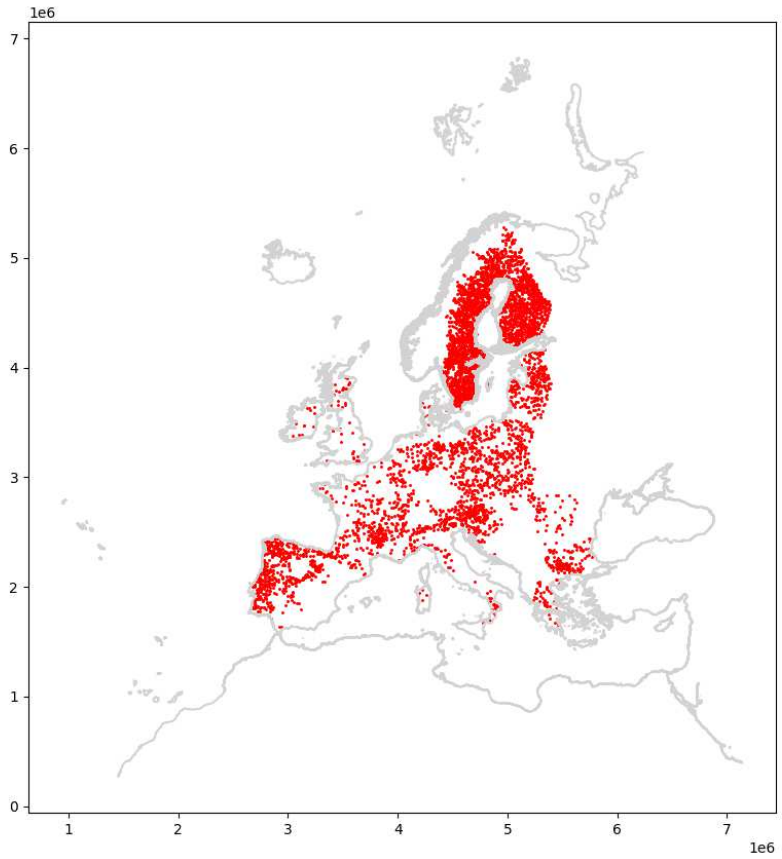


Figure 33, plot of all the points that have pH<6 and are Woodlands

With a slight increase in complexity (by incrementing the number of parameters the data needs to be filtered on), the results had a noticeable drop in quality. Some models managed to draw the desired map perfectly (starcoder, solar, gradient and llama), some didn't display the Europe borders (mistral2 and openhermes) and some couldn't generate a working code at all (zephyr and mistral3) because of dataframe joining. The reason why these dataframes cannot be joined, then filtered and then plotted is simply because they don't have any shared columns that can serve as keys for the join. Another thing that was tried during development on this example was to pass the names of columns of both dataframes to the llm so that the llm can figure out that the join isn't possible on its own. In the end, that idea hasn't worked.

Geo 4

Models\Metrics	meteor	rouge r	rouge p	rouge f	bleu	cossim	AST	levenshtein
gradient	0,3387	0,5455	0,3429	0,4211	0,1867	0,8912	0	279
llama	0,4589	0,6538	0,4857	0,5574	0,297	0,9355	0	218
openhermes	0,3726	0,5	0,4286	0,4615	0,2146	0,8326	0	220
mistral3	0,4121	0,6538	0,4857	0,5574	0,2383	0,9485	0	253
starcoder	0,635	0,75	0,6857	0,7164	0,4651	0,9571	0	116
solar	0,4403	0,4286	0,5143	0,4675	0,2685	0,8289	0	373

mistral2	0,4673	0,3878	0,5429	0,4524	0,2278	0,8185	0	478
zephyr	0,3862	0,4375	0,4	0,4179	0,229	0,9031	0	333

User Query: **Perform KMeans clustering on the TH_LAT and TH_LONG data to identify 3 clusters and plot them on a map. save it as a png.**

Here is the reference code:

```
import pandas as pd
import geopandas as gpd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

gdf = gpd.read_file('/home/fkriskov/diplomski/datasets/geo_dataframe.shp')

kmeans = KMeans(n_clusters=3)
gdf['cluster'] = kmeans.fit_predict(gdf[['TH_LAT', 'TH_LONG']])

plt.figure(figsize=(10, 10))
base =
gpd.read_file('/home/fkriskov/diplomski/datasets/Europe_coastline_shapefile
/Europe_coastline.shp')
base.plot(ax=plt.gca(), color='white', edgecolor='black')
gdf.plot(ax=plt.gca(), marker='.', column='cluster', cmap='viridis',
legend=True)

plt.title('KMeans Clustering of GeoDataFrame')
plt.savefig('clusters.png')
```

The correct answer:

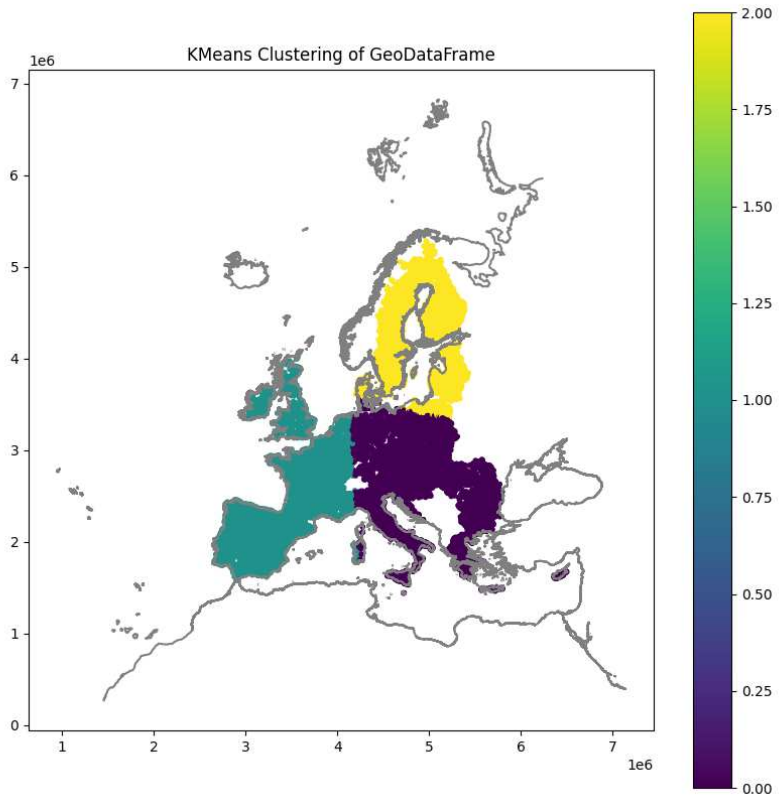


Figure 34, three-point cluster of all points from the dataset

With this example begins a set of tasks that proved to be more challenging for most LLMs as this particular example was only succeeded by two models, starcoder and llama. As before, these two models generated a correct graph, but without the borders, while all other models failed to generate any working code. The other models either generated a blank map, or some code that display all the point in the same colour, or, most commonly, had some sort of errors in the code execution itself.

Geo 5

Models\Metrics	meteor	rouge r	rouge p	rouge f	bleu	cossim	AST	levenshtein
gradient	0,5265	0,625	0,5682	0,5952	0,2373	0,9171	0	314
llama	0,501	0,75	0,5455	0,6316	0,2707	0,9231	0	298
openhermes	0,3088	0,35	0,3182	0,3333	0,2287	0,7423	0	528
mistral3	0,4192	0,5526	0,4773	0,5122	0,2045	0,8107	0	403
starcoder	0,4444	0,6	0,4773	0,5316	0,2864	0,9313	0	311
solar	0,218	0,2979	0,3182	0,3077	0,0906	0,8226	0	490
mistral2	0,413	0,6562	0,4773	0,5526	0,2396	0,9173	0	314
zephyr	0,3502	0,3409	0,3409	0,3409	0,2399	0,7626	0	426

User Query: **Create a map with markers for all locations where 'K' is above its median value, in Europe. use geopandas. save the result as a png.**

Here is the reference code:

```
import geopandas as gpd
import pandas as pd
import matplotlib.pyplot as plt

geo_df =
gpd.read_file('/home/fkriskov/diplomski/datasets/geo_dataframe.shp')
europe_df =
gpd.read_file('/home/fkriskov/diplomski/datasets/Europe_coastline_shapefile
/Europe_coastline.shp')

median_k = geo_df['K'].median()
locations = geo_df[geo_df['K'] > median_k]

fig, ax = plt.subplots(figsize=(10, 10))
europe_df.plot(ax=ax, color='gray')
locations.plot(ax=ax, marker='.', color='red')
plt.savefig('plot.png')
```

The correct answer:

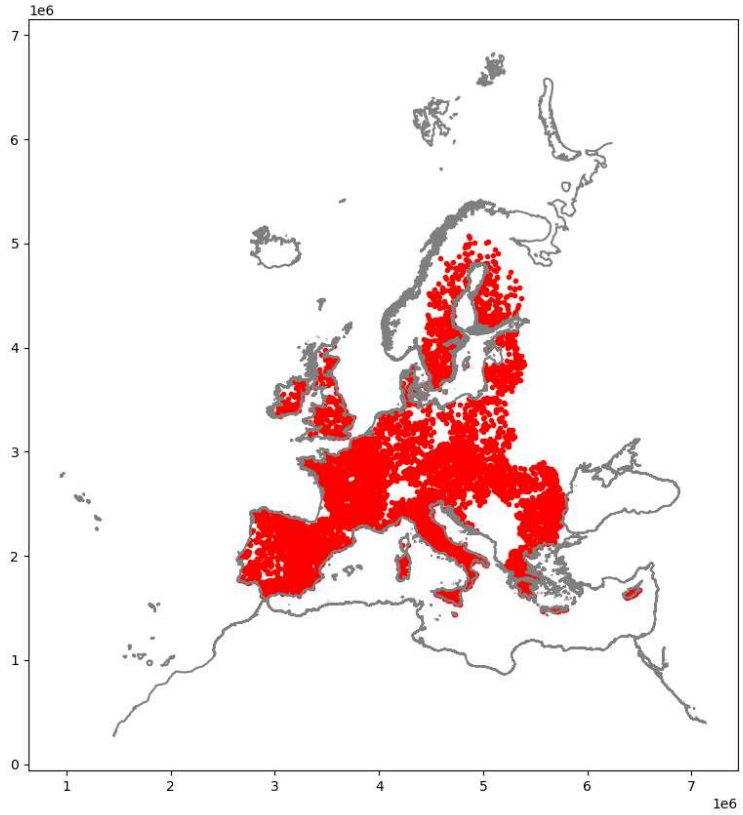


Figure 35, plot of all the points with value 'K' above its median value

Much like the previous example, the only successful models were starcoder, llama and openhermes. Other models failed to generate anything, even though they were close to the solution. Here is solar's code as an example:

```
import geopandas as gpd
import pandas as pd
import matplotlib.pyplot as plt
from shapely.geometry import Point

geo_dataframe =
gpd.read_file('/home/fkriskov/diplomski/datasets/Europe_coastline_shapefile
/Europe_coastline.shp')

europe_coastline =
gpd.read_file('/home/fkriskov/diplomski/datasets/geo_dataframe.shp')

median_k = pd.Series(geo_dataframe['K']).median()

above_median_k = geo_dataframe['K'] > median_k

marker_points = geo_dataframe.loc[above_median_k, 'POINTID'].apply(Point)

fig, ax = plt.subplots(figsize=(10,10))
europe_coastline.plot(ax=ax, color='None', edgecolor='black')
marker_points.plot(ax=ax, markersize=10, marker='.')

plt.savefig('plot.png')
```

This code is very similar to the reference code, which gives high scores on some metrics like rouge and cosine similarity, but it has some obvious mistakes. For example, it loaded the files into wrong variables and while filtering the dataframe values above the median, didn't return them. Also the line with the marker_points is completely unnecessary. These errors would be obvious to someone familiar with programming and graph plotting, but in this case, it's not acceptable as the users this system is designed for, wouldn't know what's wrong, nor would they know how to fix it. This is what the result would be if these mistakes were fixed:

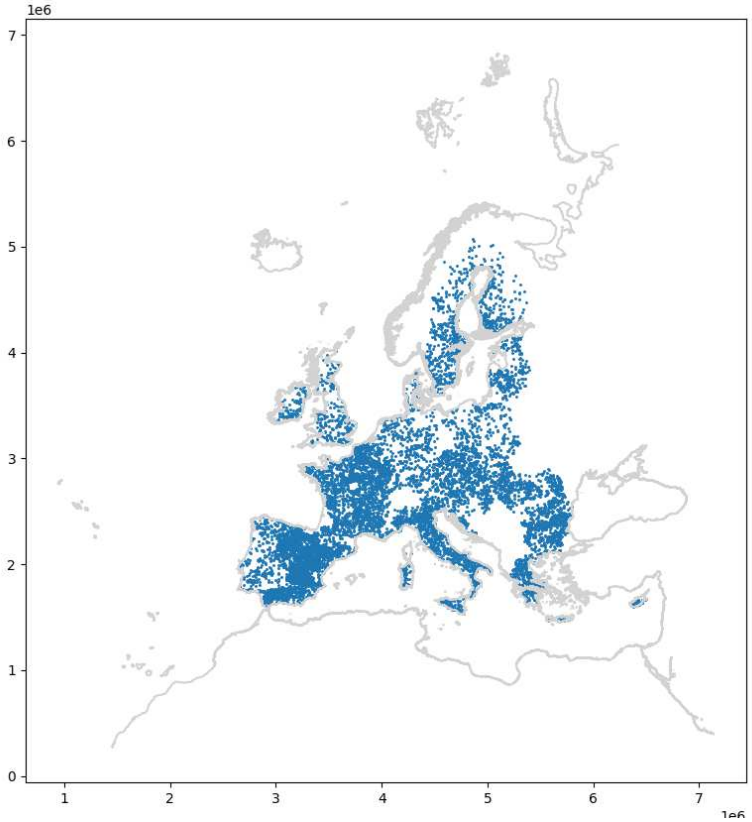


Figure 36, geo 5 result for fixed solar code

Geo 6

Models\Metrics	meteor	rouge r	rouge p	rouge f	bleu	cossim	AST	levenshtein
gradient	0,2725	0,5172	0,3191	0,3947	0,1207	0,8767	0	413
llama	0,3203	0,4571	0,3404	0,3902	0,1461	0,8675	0	384
openhermes	0,2842	0,4571	0,3404	0,3902	0,1054	0,8365	0	391
mistral3	0,2794	0,4571	0,3404	0,3902	0,1211	0,8835	0	479
starcoder	0,3289	0,5882	0,4255	0,4938	0,1628	0,9122	0	345
solar	0,2601	0,35	0,2979	0,3218	0,1259	0,8546	0	494
mistral2	0,299	0,4146	0,3617	0,3864	0,1463	0,8978	0	450
zephyr	0,2704	0,28	0,2979	0,2887	0,1234	0,8312	0	624

User Query: **Generate a heatmap where each point is weighted by 'pH_CaCl2', in Europe. Don't merge these shapefiles just plot them. use geopandas. save the result as a png.**

Here is the reference code:

```
import pandas as pd
import geopandas as gpd
import matplotlib.pyplot as plt

gdf = gpd.read_file('/home/fkriskov/diplomski/datasets/geo_dataframe.shp')

europe_shapefile =
gpd.read_file('/home/fkriskov/diplomski/datasets/Europe_coastline_shapefile
/Europe_coastline.shp')

fig, ax = plt.subplots(figsize=(10, 10))
europe_shapefile.plot(ax=ax, color='white', edgecolor='black')

gdf.plot(ax=ax, marker='.', column='pH_CaCl2', cmap='hot', legend=True,
markersize=gdf['pH_CaCl2'])

plt.title('Heatmap of pH_CaCl2 in Europe')
plt.savefig('plot.png')
```

The correct answer:

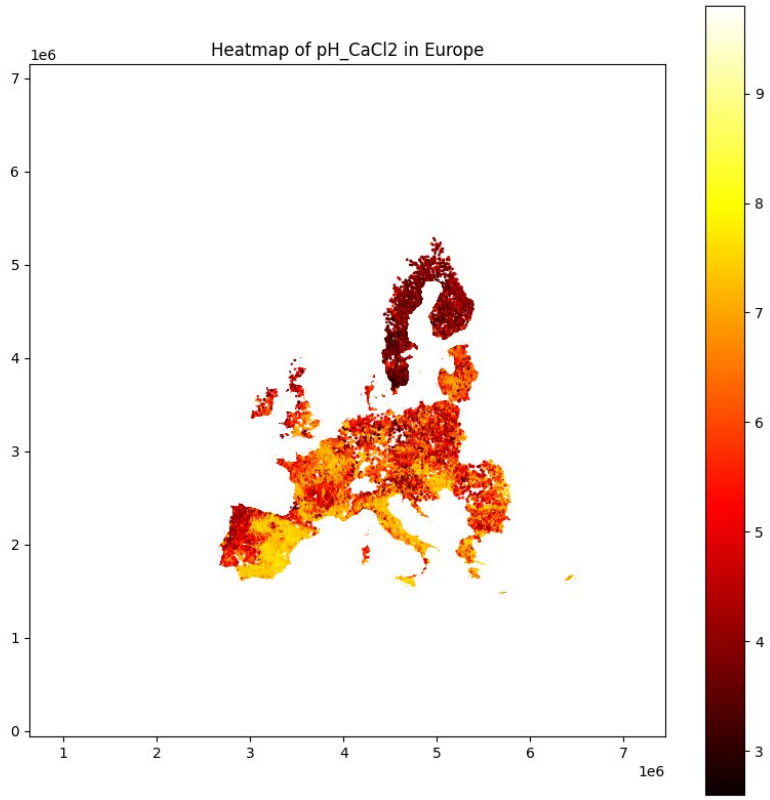


Figure 37, heatmap of pH of all the points

This example had the same low success rate despite being a simple request. The key parts here were plotting the Europe border, and then plotting the pH data. Some models tried to unnecessarily generate their own colour scheme which didn't work, openhermes' code was one line away from the correct solution, which explains his very high metric scores. Here, only starcoder and mistral3 generated a correct heatmap.

Geo 7

Models\Metrics	meteor	rouge r	rouge p	rouge f	bleu	cossim	AST	levenshtein
gradient	0,5169	0,5172	0,5556	0,5357	0,3038	0,8648	0	225
llama	1	1	1	1	1	1	1	0
openhermes	0,4886	0,2545	0,5185	0,3415	0,1617	0,9054	0	387
mistral3	0,5647	0,625	0,5556	0,5882	0,3706	0,9327	0	223
starcoder	0,4491	0,2222	0,5185	0,3111	0,1373	0,8925	0	568
solar	0,3136	0,3636	0,4444	0,4	0,2255	0,813	0	313
mistral2	0,4463	0,2381	0,5556	0,3333	0,1331	0,9017	0	458
zephyr	0,491	0,3158	0,6667	0,4286	0,1117	0,7683	0	441

User Query: **Create a map with markers for points where 'K' is in the top 10 percentile, in Europe. Don't merge these shapefiles just plot them. use geopandas. save the result as a png.**

Here is the reference code:

```
import pandas as pd
import geopandas as gpd
import matplotlib.pyplot as plt

gdf = gpd.read_file('/home/fkriskov/diplomski/datasets/geo_dataframe.shp')

top_10_percentile = gdf['K'].quantile(0.90)

top_k_points = gdf[gdf['K'] > top_10_percentile]

europe_shapefile =
gpd.read_file('/home/fkriskov/diplomski/datasets/Europe_coastline_shapefile
/Europe_coastline.shp')

fig, ax = plt.subplots(figsize=(10, 10))
europe_shapefile.plot(ax=ax, color='gray', edgecolor='black')

top_k_points.plot(ax=ax, marker='.', color='red', markersize=5)

plt.title('Top 10 Percentile of K Values in Europe')
plt.savefig('plot.png')
```

The correct answer:

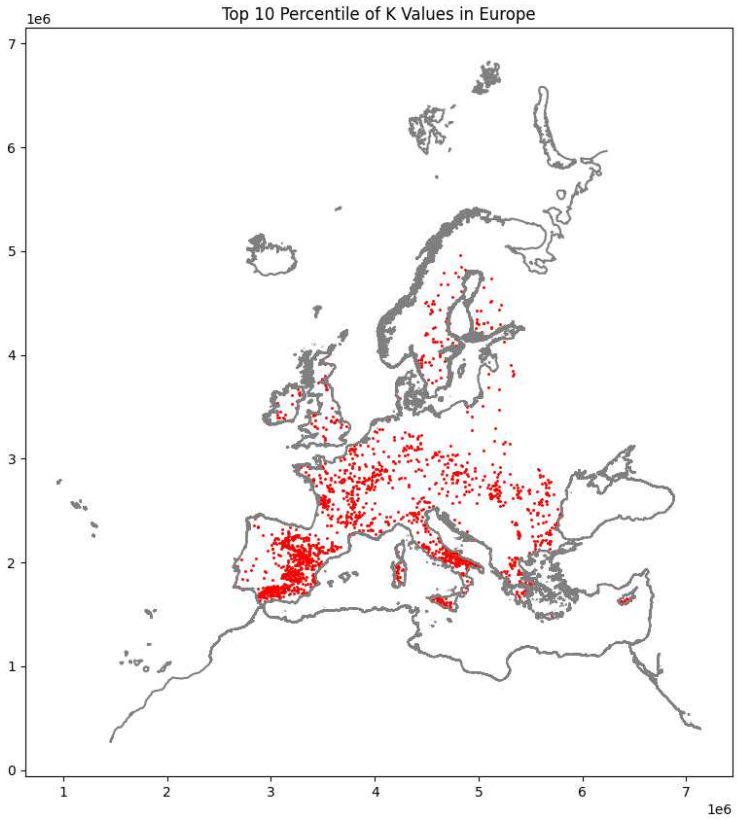


Figure 38, plot of all the points where value 'K' is in the top 10 percentile

Same as Geo 5 example, only llama and starcoder managed to generate the correct maps for this example. All these examples follow the same three step solution recipe; load the data, filter the data by the given parameters and then plot that data, but models still manage to solve some tasks and fail others.

Geo 8

Models\Metrics	meteor	rouge r	rouge p	rouge f	bleu	cossim	AST	levenshtein
gradient	0,2725	0,5172	0,3191	0,3947	0,1207	0,8767	0	413
llama	0,3203	0,4571	0,3404	0,3902	0,1461	0,8675	0	384
openhermes	0,2842	0,4571	0,3404	0,3902	0,1054	0,8365	0	391
mistral3	0,2794	0,4571	0,3404	0,3902	0,1211	0,8835	0	479
starcoder	0,3289	0,5882	0,4255	0,4938	0,1628	0,9122	0	345
solar	0,2601	0,35	0,2979	0,3218	0,1259	0,8546	0	494
mistral2	0,299	0,4146	0,3617	0,3864	0,1463	0,8978	0	450
zephyr	0,2704	0,28	0,2979	0,2887	0,1234	0,8312	0	624

User Query: **Plot clusters of points with 'pH_H2O'>5 and 'pH_H2O'<5 in Europe.**

Here is the reference code:

```
import geopandas as gpd
import matplotlib.pyplot as plt

gdf_points =
gpd.read_file('/home/fkriskov/diplomski/datasets/geo_dataframe.shp')

gdf_europe =
gpd.read_file('/home/fkriskov/diplomski/datasets/Europe_coastline_shapefile
/Europe_coastline.shp')

high_ph = gdf_points[gdf_points['pH_H2O'] > 5]
low_ph = gdf_points[gdf_points['pH_H2O'] < 5]

fig, ax = plt.subplots(figsize=(10, 10))

gdf_europe.plot(ax=ax, color='lightgrey', edgecolor='black')
high_ph.plot(ax=ax, marker='.', color='blue', markersize=5, label='pH_H2O >
5')
low_ph.plot(ax=ax, marker='.', color='red', markersize=5, label='pH_H2O <
5')

plt.legend()
plt.title('Clusters of Points with pH_H2O > 5 and pH_H2O < 5 in Europe')

plt.savefig('ph_h2o_clusters.png', dpi=300, bbox_inches='tight')
```

The correct answer:

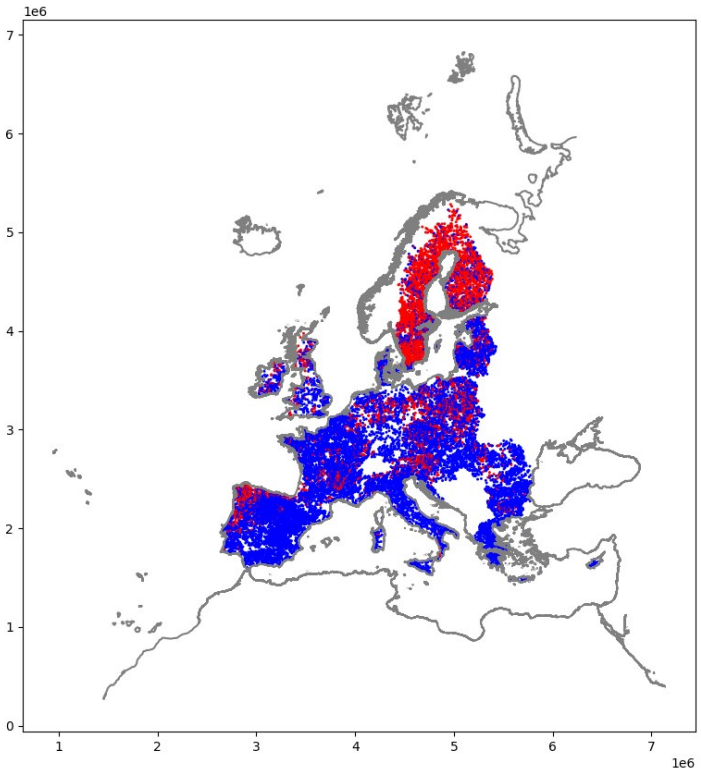


Figure 39, correct solution for Geo 8

This example required the models to plot the same parameter on two different criteria, with different colours. This is on par with the previous examples which require data filtering followed by data plotting. This task was successfully completed by Starcoder, Gradient and Llama, while other models either encountered an execution error or tried merging the dataframes which resulted in blank graphs. In this example, Llama's solution was only partial:

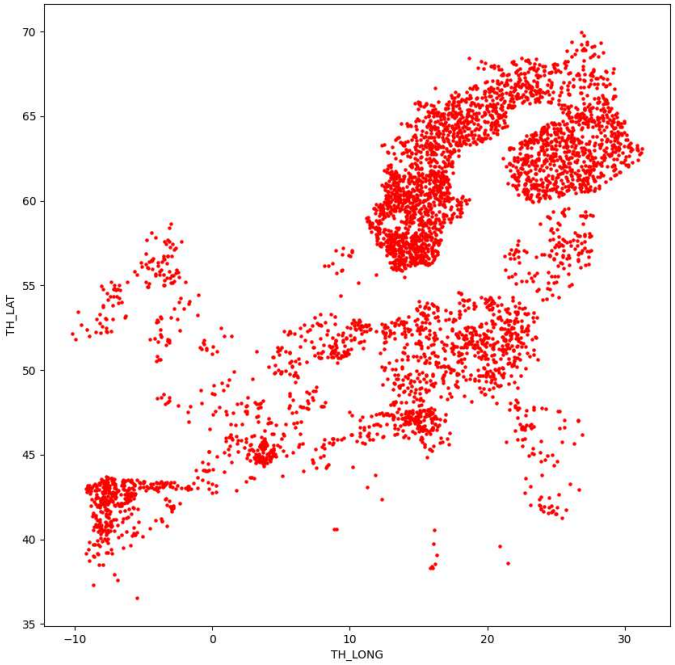


Figure 40, Llama solution for Geo 8

Geo 9

Models\Metrics	meteor	rouge r	rouge p	rouge f	bleu	cossim	AST	levenshtein
gradient	0,5169	0,5172	0,5556	0,5357	0,3038	0,8648	0	225
llama	1	1	1	1	1	1	1	0
openhermes	0,4886	0,2545	0,5185	0,3415	0,1617	0,9054	0	387
mistral3	0,5647	0,625	0,5556	0,5882	0,3706	0,9327	0	223
starcoder	0,4491	0,2222	0,5185	0,3111	0,1373	0,8925	0	568
solar	0,3136	0,3636	0,4444	0,4	0,2255	0,813	0	313
mistral2	0,4463	0,2381	0,5556	0,3333	0,1331	0,9017	0	458
zephyr	0,491	0,3158	0,6667	0,4286	0,1117	0,7683	0	441

User Query: **Create a map displaying the distribution of soil types ('LC0_Desc') across Europe. Each soil type should be represented by a different colour. Use geopandas and save the map as a png.**

Here is the reference code:

```
import geopandas as gpd
import matplotlib.pyplot as plt

df = gpd.read_file('/home/fkriskov/diplomski/datasets/geo_dataframe.shp')

europe =
gpd.read_file('/home/fkriskov/diplomski/datasets/Europe_coastline_shapefile
/Europe_coastline.shp')

fig, ax = plt.subplots(figsize=(10,10))
europe.plot(ax=ax, color='lightblue', edgecolor='black')

df.plot(ax=ax, column='LC0_Desc', categorical=True, markersize=5,
marker='.', legend=True)

plt.savefig('soil_types_map.png')
```

The correct answer:

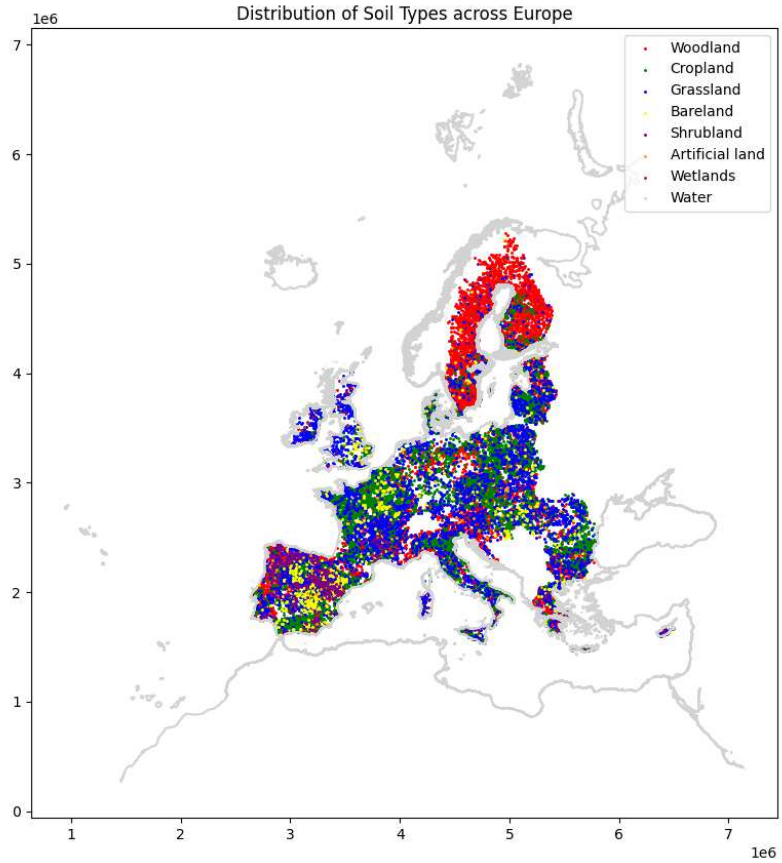


Figure 41, correct solution for Geo 9

Same as the previous example, the models that were successful are Starcoder, Gradient and Llama, while other models failed to generate anything. Here is how Gradient and Llama graphs looked like:

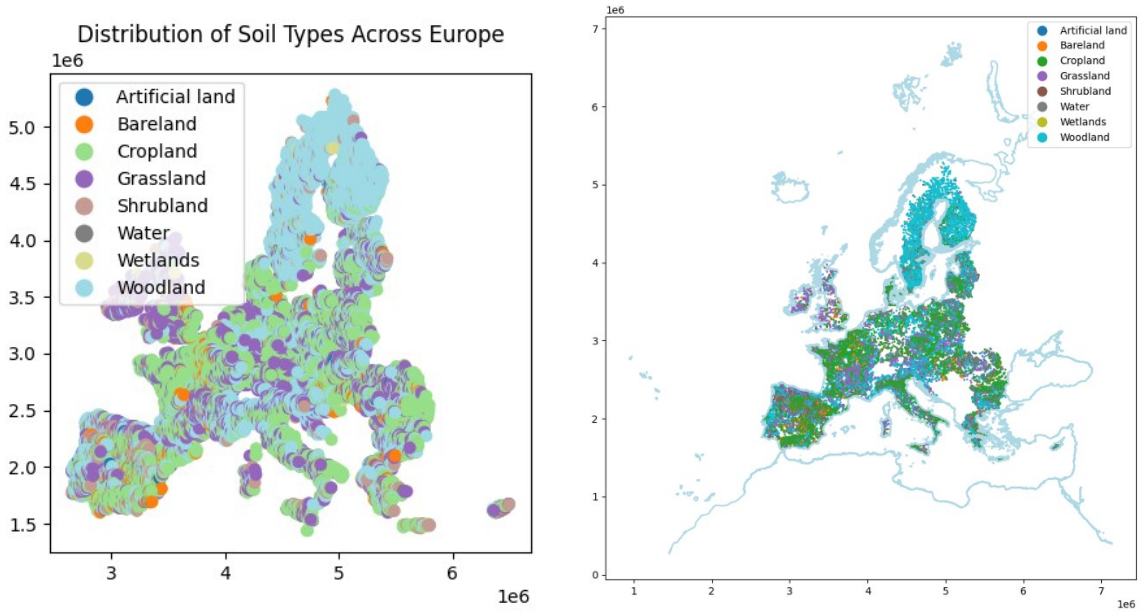


Figure 42, Gradient and Llama solutions for Geo 9

Geo 10

Models\Metrics	meteor	rouge r	rouge p	rouge f	bleu	cossim	AST	levenshtein
gradient	0,5169	0,5172	0,5556	0,5357	0,3038	0,8648	0	225
llama	1	1	1	1	1	1	1	0
openhermes	0,4886	0,2545	0,5185	0,3415	0,1617	0,9054	0	387
mistral3	0,5647	0,625	0,5556	0,5882	0,3706	0,9327	0	223
starcoder	0,4491	0,2222	0,5185	0,3111	0,1373	0,8925	0	568
solar	0,3136	0,3636	0,4444	0,4	0,2255	0,813	0	313
mistral2	0,4463	0,2381	0,5556	0,3333	0,1331	0,9017	0	458
zephyr	0,491	0,3158	0,6667	0,4286	0,1117	0,7683	0	441

User Query: **Plot all the LC0_Desc='Grassland' and LC0_Desc='Woodland' points where 'OC'>20. Use geopandas and save the map as a png.**

Here is the reference code:

```
import geopandas as gpd
import matplotlib.pyplot as plt

geo_dataframe =
gpd.read_file('/home/fkriskov/diplomski/datasets/geo_dataframe.shp')
europe_shapefile =
gpd.read_file('/home/fkriskov/diplomski/datasets/Europe_coastline_shapefile
/Europe_coastline.shp')

filtered_geo_dataframe = geo_dataframe[(geo_dataframe['LC0_Desc'] ==
'Grassland') | (geo_dataframe['LC0_Desc'] == 'Woodland')]
filtered_geo_dataframe =
filtered_geo_dataframe[filtered_geo_dataframe['OC'] > 20]
```

```
fig, ax = plt.subplots(figsize=(10, 10))
europe_shapefile.plot(ax=ax, color='gray')
filtered_geo_dataframe.plot(ax=ax, marker='.', color='red')
plt.savefig('plot.png')
```

The correct answer:

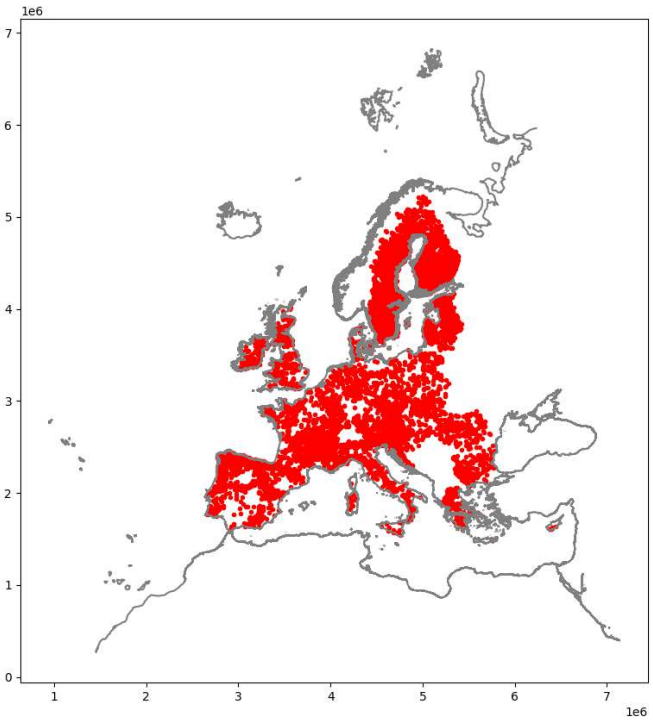


Figure 43, correct solution for Geo 10

In this final geo-spatial example, along with Starcoder, Gradient and Llama, the correct result was also provided by Zephyr. All four models had very similar codes, and executed all the aspects of correctly, while the other models failed to generate anything. Here are some examples of other models' solutions:

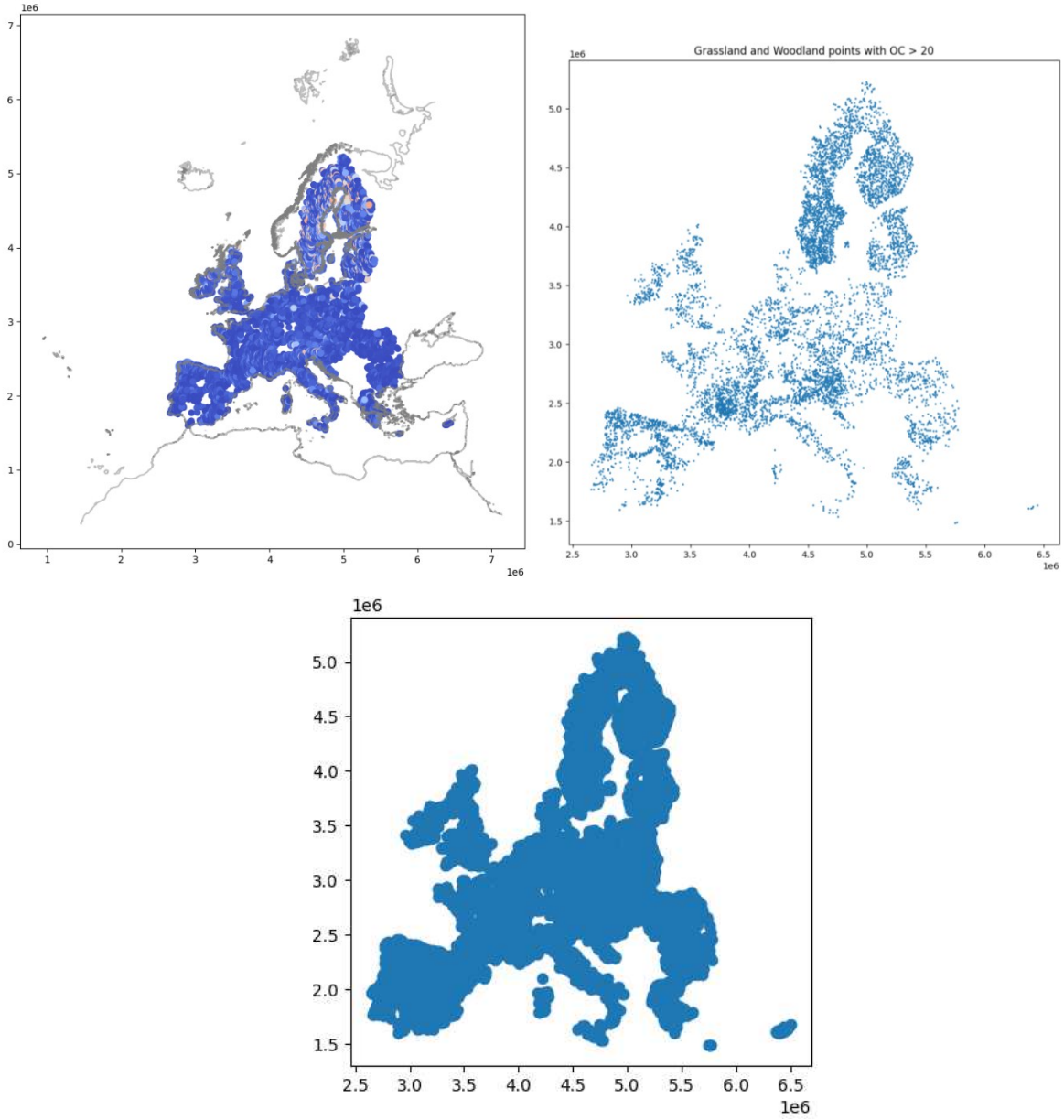


Figure 44, Zephyr, Llama and Gradient solutions for Geo 10

Infer 1

Models\Metrics	meteor	rouge r	rouge p	rouge f	bleu	cossim	AST	levenshtein
gradient	0,4274	0,64	0,4706	0,5424	0,2177	0,8308	0	359
llama	0,4907	0,5405	0,5882	0,5634	0,2587	0,8625	0	257
openhermes	0,1187	0,25	0,2059	0,2258	0,0691	0,8293	0	279
mistral3	0,4164	0,4286	0,4412	0,4348	0,2282	0,9102	0	260
starcoder	1	1	1	1	1	1	1	0
solar	0,4714	0,5714	0,5882	0,5797	0,3159	0,9491	0	324
mistral2	0,1459	0,4074	0,3235	0,3607	0,0803	0,8505	0	288
zephyr	0,4958	0,5135	0,5588	0,5352	0,3007	0,9239	0	238

User Query: **Is there a significant relationship between land type (LC0_Desc) and pH_H2O? Use chi square from scipy.**

Here is the reference code:

```
import pandas as pd
from scipy.stats import chi2_contingency

# Load the dataset
df = pd.read_csv('/home/fkriskov/diplomski/datasets/lucas-soil-2018-
v2/lucas-soil-2018 copy.csv')

# Create a contingency table for LC0_Desc and pH_H2O
contingency_table = pd.crosstab(df['LC0_Desc'], df['pH_H2O'])

# Perform the chi-square test
chi2, p, _, _ = chi2_contingency(contingency_table)

# Print the results
print("Chi2:", chi2)
print("P-value:", p)
```

The correct answer: 13418.924895244883

This was the first example of an inferential statistics question, which all but two models calculated correctly. The incorrect solutions were by Gradient and Mistral2:

```
df['LC0_Desc'] = df['LC0_Desc'].astype('category')
df['pH_H2O'] = df['pH_H2O'].astype(float)
chi2, p, dof, expected = stats.chisquare(df['pH_H2O'], df['LC0_Desc'])


chi2, pvalue, dof, expected =
stats.chi2_contingency(pd.get_dummies(df['LC0_Desc']).values,
pd.get_dummies(df['pH_H2O'] > 5).values)
```

Infer 2

Models\Metrics	meteor	rouge r	rouge p	rouge f	bleu	cossim	AST	levenshtein
gradient	0,4079	0,5909	0,52	0,5532	0,2162	0,8547	0	163
llama	0,7071	0,7826	0,72	0,75	0,4895	0,9646	0	83
openhermes	0,5004	0,5	0,48	0,4898	0,3665	0,8982	0	142
mistral3	0,735	0,7826	0,72	0,75	0,5912	0,9733	0	71
starcoder	0,7417	0,6552	0,76	0,7037	0,5072	0,9742	0	66
solar	0,5344	0,6522	0,6	0,625	0,4419	0,9576	0	119
mistral2	0,3318	0,5	0,48	0,4898	0,2003	0,9207	0	161
zephyr	0,5067	0,3824	0,52	0,4407	0,3147	0,8705	0	194

User Query: **Is there a significant difference between 'N' in Austria and France? Use ANOVA from scipy.**

Here is the reference code:

```
import pandas as pd
from scipy.stats import f_oneway
```

```
file_path = '/home/fkriskov/diplomski/datasets/lucas-soil-2018-v2/lucas-
soil-2018 copy.csv'
df = pd.read_csv(file_path)

austria_N = df[df['NUTS_0'] == 'AT']['N']
france_N = df[df['NUTS_0'] == 'FR']['N']

anova_result = f_oneway(austria_N.dropna(), france_N.dropna())

print("F-statistic:", anova_result.statistic)
print("p-value:", anova_result.pvalue)
```

The correct answer: F-statistic: 257.14892962811507, p-value: 1.0339087114935148e-55

This was another successful task, except for gradient. Gradient tried to use statsmodels module which wasn't installed, but even with the module available, it gives an execution error.

```
import pandas as pd
import statsmodels.api as sm

df = pd.read_csv('/home/fkriskov/diplomski/datasets/lucas-soil-2018-
v2/lucas-soil-2018 copy.csv')

grouped_df = df.groupby('NUTS_0')['pH_CaCl2'].mean()

anova_results = sm.stats.anova_lm(grouped_df)

print(anova_results)
```

Infer 3

Models\Metrics	meteor	rouge r	rouge p	rouge f	bleu	cossim	AST	levenshtein
gradient	0,2354	0,4211	0,2759	0,3333	0,0798	0,7796	0	232
llama	0,4361	0,4333	0,4483	0,4407	0,1914	0,85	0	186
openhermes	0,4808	0,6087	0,4828	0,5385	0,1988	0,9329	0	157
mistral3	0,4117	0,5652	0,4483	0,5	0,1479	0,8732	0	180
starcoder	0,2754	0,3571	0,3448	0,3509	0,1015	0,7875	0	203
solar	0,4741	0,4643	0,4483	0,4561	0,2458	0,9336	0	147
mistral2	0,4042	0,3548	0,3793	0,3667	0,1441	0,8541	0	178
zephyr	0,2191	0,1765	0,3103	0,225	0,0519	0,8343	0	352

User Query: **Which parameter has the strongest correlation with EC among {pH_CaCl2, pH_H2O, OC, CaCO3, P, N, K}?**

Here is the reference code:

```
import pandas as pd

file_path = '/home/fkriskov/diplomski/datasets/lucas-soil-2018-v2/lucas-
soil-2018 copy.csv'
df = pd.read_csv(file_path)

parameters = ['pH_CaCl2', 'pH_H2O', 'OC', 'CaCO3', 'P', 'N', 'K']

correlations = df[parameters + ['EC']].corr()['EC'].drop('EC')
```

```
strongest_correlation = correlations.idxmax()
correlation_value = correlations.max()

print("Parameter with the strongest correlation with EC:",
strongest_correlation)
print("Correlation value:", correlation_value)
```

The correct answer:

Parameter with the strongest correlation with EC: pH_CaCl2

Correlation value: 0.217453954161236

This example was correctly solved by only starcoder and gradient. The two main ways to solve this problem was to either generate a full correlation matrix, drop the 'EC' row, and find the argmax of the matrix with the 'EC' parameter, or, to directly calculate the correlations with the 'EC':

```
import pandas as pd

df = pd.read_csv('/home/fkriskov/diplomski/datasets/lucas-soil-2018-
v2/lucas-soil-2018 copy.csv')
parameters = ['pH_CaCl2', 'pH_H2O', 'OC', 'CaCO3', 'P', 'N', 'K']
correlations = df[parameters].corrwith(df['EC'])
strongest_correlation = correlations.abs().idxmax()
print(f"The parameter with the strongest correlation with EC is
{strongest_correlation}.")
```

Other models either didn't exclude the correlation of 'EC' with itself, which resulted in the maximum correlation to be 1, or failed to generate the correct code, which resulted in the wrong parameter as the answer.

Infer 4

Models\Metrics	meteor	rouge r	rouge p	rouge f	bleu	cossim	AST	levenshtein
gradient	0,5055	0,6333	0,4872	0,5507	0,2568	0,9456	0	194
llama	0,3114	0,48	0,3077	0,375	0,1105	0,8772	0	270
openhermes	0,3257	0,5417	0,3333	0,4127	0,1382	0,7851	0	279
mistral3	0,2647	0,55	0,2821	0,3729	0,0798	0,7941	0	309
starcoder	0,5628	0,6667	0,5641	0,6111	0,3937	0,9542	0	160
solar	0,3376	0,3939	0,3333	0,3611	0,1917	0,836	0	255
mistral2	0,3257	0,5	0,3333	0,4	0,1382	0,8205	0	250
zephyr	0,398	0,3784	0,359	0,3684	0,2219	0,7825	0	252

User Query: **Perform a t-test to compare 'K' between Grassland and Cropland.**

Here is the reference code:

```
import pandas as pd
from scipy.stats import ttest_ind

df = pd.read_csv('/home/fkriskov/diplomski/datasets/lucas-soil-2018-
v2/lucas-soil-2018 copy.csv')

Grassland = df[df['LC0_Desc'] == 'Grassland']['K']
Cropland = df[df['LC0_Desc'] == 'Cropland']['K']
```

```
t_statistic, p_value = ttest_ind(Grassland, Cropland)

print(f"T-statistic: {t_statistic}, p-value: {p_value}")
```

The correct answer: T-statistic: -9.99983103191002, p-value: 1.9061232022286383e-23

All models managed to correctly calculate this t-test. Starcoder once again had the best solution.

Infer 5

Models\Metrics	meteor	rouge r	rouge p	rouge f	bleu	cossim	AST	levenshtein
gradient	0,1437	0,3636	0,24	0,2892	0,118	0,5755	0	382
llama	0,2167	0,5714	0,32	0,4103	0,1307	0,686	0	372
openhermes	0,3306	0,5116	0,44	0,4731	0,2227	0,6526	0	308
mistral3	0,3211	0,5641	0,44	0,4944	0,1777	0,6325	0	376
starcoder	0,1139	0,3095	0,26	0,2826	0,1136	0,667	0	414
solar	0,0599	0,35	0,14	0,2	0,0145	0,4255	0	460
mistral2	0,2643	0,3962	0,42	0,4078	0,253	0,6603	0	426
zephyr	0,2274	0,5667	0,34	0,425	0,1644	0,6353	0	340

User Query: **Plot a linear regression analysis to see the relationship between 'pH_H2O' and 'K'.**

Here is the reference code:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
import numpy as np

df = pd.read_csv('/home/fkriskov/diplomski/datasets/lucas-soil-2018-
v2/lucas-soil-2018 copy.csv')

X = df[['pH_H2O']].values
y = df['K'].values

model = LinearRegression()
model.fit(X, y)

print(f"Intercept: {model.intercept_}, Coefficient: {model.coef_[0]}")

sns.scatterplot(x='sepal_width', y='petal_width', data=df)
plt.plot(df['pH_H2O'], model.predict(X), color='red')
plt.title('Linear Regression: pH_H2O vs K')
plt.savefig('linreg.png')
```

The correct answer:

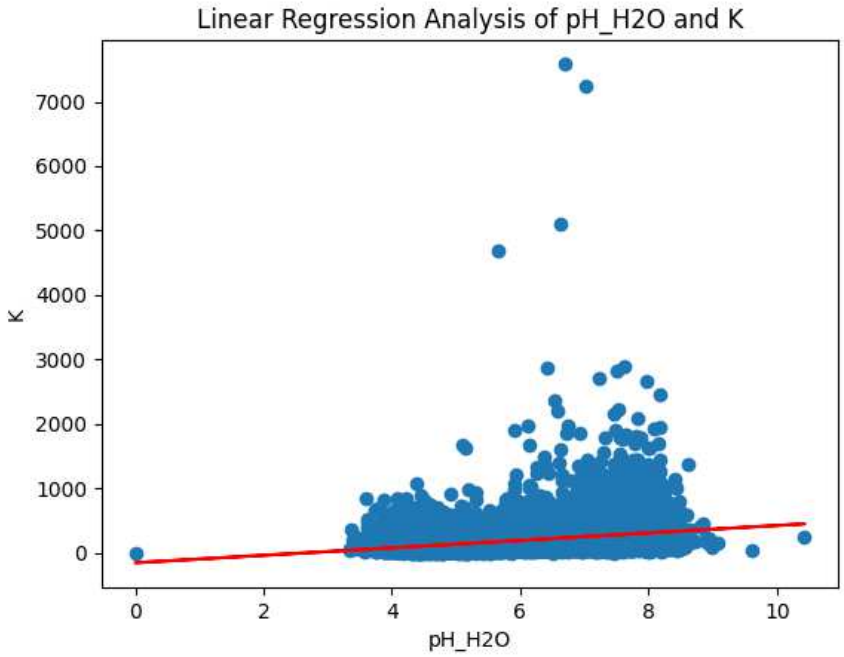


Figure 45, Linear regression between pH_H2O and K

With the geo-spatial examples, the goal was to check models' map drawing capabilities by understanding geographical coordinates and data filtering. This example's aim was to check the models' capabilities in drawing scatter graphs with some meaningful information about the data shown. Linear regression is a very common, and very useful machine learning algorithm, that shows the relationship between two variables by fitting a linear equation to the observed data. Scatter plots with linear regression lines help in visualizing how well the model fits the data and in understanding the nature of the relationship between the variables, such as its strength and direction. This is useful for tasks like predicting outcomes, identifying trends, and making data-driven decisions.

In this example, gradient, mistral3, solar and mistral2 were unsuccessful.

Infer 6

Models\Metrics	meteor	rouge r	rouge p	rouge f	bleu	cossim	AST	levenshtein
gradient	0,477	0,4167	0,5556	0,4762	0,2045	0,8709	0	248
llama	0,3419	0,3714	0,3611	0,3662	0,1724	0,7951	0	206
openhermes	0,3323	0,375	0,4167	0,3947	0,2031	0,8366	0	238
mistral3	0,2588	0,4375	0,3889	0,4118	0,1884	0,7675	0	237
starcoder	0,3993	0,3824	0,3611	0,3714	0,1762	0,887	0	166
solar	0,3748	0,4524	0,5278	0,4872	0,1575	0,8966	0	255
mistral2	0,2575	0,3421	0,3611	0,3514	0,1508	0,8881	0	259
zephyr	0,5202	0,3725	0,5278	0,4368	0,1784	0,8998	0	283

User Query: **Construct a 95% confidence interval for the mean 'OC' content in the dataset.**

Here is the reference code:

```
import pandas as pd
import scipy.stats as stats
import numpy as np

df = pd.read_csv('/home/fkriskov/diplomski/datasets/lucas-soil-2018-
v2/lucas-soil-2018 copy.csv')

mean_oc = df['OC'].mean()
sem_oc = stats.sem(df['OC'], nan_policy='omit')

confidence_interval = stats.t.interval(0.95, len(df['OC'].dropna()) - 1,
loc=mean_oc, scale=sem_oc)

print("95% Confidence Interval for mean 'OC':", confidence_interval)
```

The correct answer: 95% Confidence Interval for mean 'OC': (46.356662130276014, 48.67837263584283)

Looking at the metrics table for this example, it could be concluded that the results are very similar across all models, which is only partially true since Starcoder, Gradient, Llama and Mistral3 managed to calculate the correct solutions, while the other models generated seemingly correct codes which didn't calculate the interval correctly. Here is the code that Zephyr generated, that had the highest cosine similarity to the reference code, but was incorrect:

```
import pandas as pd
import numpy as np
import scipy.stats as stats

df = pd.read_csv('/home/fkriskov/diplomski/datasets/lucas-soil-2018-
v2/lucas-soil-2018 copy.csv')
oc_values = df['OC'].values
n, mean_oc, var_oc = np.size(oc_values), np.mean(oc_values),
np.var(oc_values)
std_error = np.sqrt(var_oc / n)
confidence_level = 0.95
t_value = stats.t.ppf(confidence_level / 2, n-1)
lower_bound = mean_oc - t_value * std_error
upper_bound = mean_oc + t_value * std_error
print(f'95% confidence interval for mean "OC" content: [{lower_bound:.2f},
{upper_bound:.2f}]')
```

Infer 7

Models\Metrics	meteor	rouge r	rouge p	rouge f	bleu	cossim	AST	levenshtein
gradient	0,4274	0,64	0,4706	0,5424	0,2177	0,8308	0	359
llama	0,4907	0,5405	0,5882	0,5634	0,2587	0,8625	0	257
openhermes	0,1187	0,25	0,2059	0,2258	0,0691	0,8293	0	279
mistral3	0,4164	0,4286	0,4412	0,4348	0,2282	0,9102	0	260
starcoder	1	1	1	1	1	1	1	0
solar	0,4714	0,5714	0,5882	0,5797	0,3159	0,9491	0	324
mistral2	0,1459	0,4074	0,3235	0,3607	0,0803	0,8505	0	288
zephyr	0,4958	0,5135	0,5588	0,5352	0,3007	0,9239	0	238

User Query: **Using the Central Limit Theorem, simulate the sampling distribution of the mean 'pH_H2O' for sample sizes of 30. Plot the distribution and compare it to the normal distribution.**

Here is the reference code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
df = pd.read_csv('/home/fkriskov/diplomski/datasets/lucas-soil-2018-
v2/lucas-soil-2018 copy.csv')

# Set the sample size
sample_size = 30
n_simulations = 1000

# Simulate the sampling distribution of the mean 'pH_H2O'
sample_means = [df['pH_H2O'].sample(sample_size, replace=True).mean() for _
in range(n_simulations)]

# Plot the sampling distribution and compare to a normal distribution
sns.histplot(sample_means, kde=True, stat='density', color='blue',
label='Sample Means')
sns.kdeplot(np.random.normal(np.mean(sample_means), np.std(sample_means),
n_simulations), color='red', label='Normal Distribution')
plt.xlabel('Mean pH_H2O')
plt.ylabel('Density')
plt.title('Sampling Distribution of the Mean pH_H2O')
plt.legend()
plt.savefig('plot.png')
plt.show()
```

The correct answer:

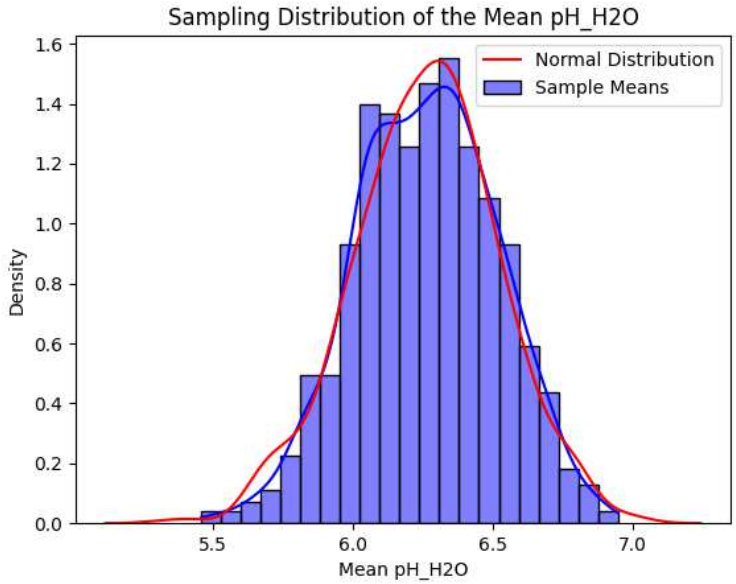


Figure 46, sample distribution of the mean pH_H2O

This example required the models to generate a graph of pH_H2O samples versus a normal distribution. Among the correct models, the best one was Starcoder, but Mistral3 had the highest scores. Except Starcoder, other correct models generated generated partially correct solutions, either containing only parts of the reference solution or resembling the reference solution. Other graphs had neither of those things, so those didn't count towards being correct:

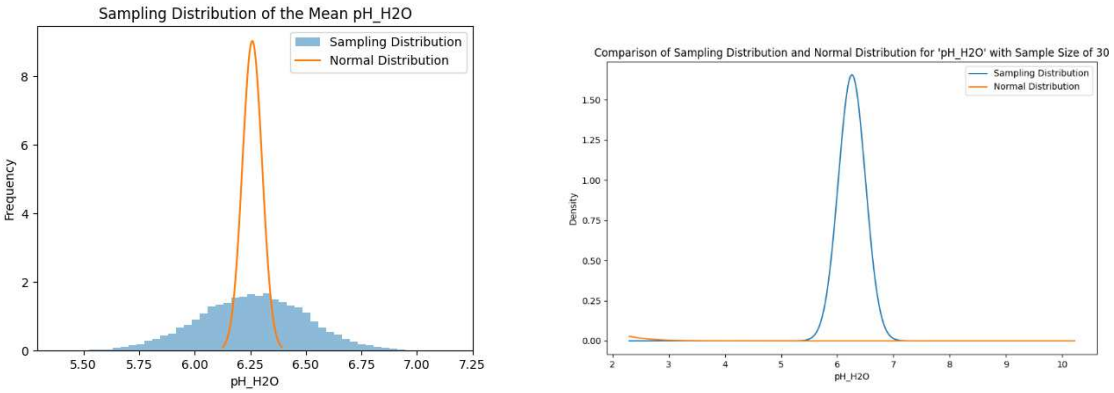


Figure 47, llama and mistral2 graphs for Infer 7

Infer 8

Models\Metrics	meteor	rouge r	rouge p	rouge f	bleu	cossim	AST	levenshtein
gradient	0,4274	0,64	0,4706	0,5424	0,2177	0,8308	0	359
llama	0,4907	0,5405	0,5882	0,5634	0,2587	0,8625	0	257
openhermes	0,1187	0,25	0,2059	0,2258	0,0691	0,8293	0	279
mistral3	0,4164	0,4286	0,4412	0,4348	0,2282	0,9102	0	260
starcoder	1	1	1	1	1	1	1	0
solar	0,4714	0,5714	0,5882	0,5797	0,3159	0,9491	0	324
mistral2	0,1459	0,4074	0,3235	0,3607	0,0803	0,8505	0	288
zephyr	0,4958	0,5135	0,5588	0,5352	0,3007	0,9239	0	238

User Query: **Calculate the z-scores for 'EC' and identify any outliers (z-score > 3 or < -3).**

Here is the reference code:

```
import pandas as pd
import scipy.stats as stats

# Load the dataset
df = pd.read_csv('/home/fkriskov/diplomski/datasets/lucas-soil-2018-
v2/lucas-soil-2018 copy.csv')

# Calculate the z-scores for 'EC'
df['EC_zscore'] = stats.zscore(df['EC'], nan_policy='omit')

# Identify outliers (z-score > 3 or < -3)
outliers = df[(df['EC_zscore'] > 3) | (df['EC_zscore'] < -3)]

# Print the outliers
print(outliers[['POINTID', 'EC', 'EC_zscore']])
```

The correct answer:

```
       POINTID      EC  EC_zscore
58    47502772  145.60   4.977129
175   46222728  123.50   4.112474
256   47482734   96.50   3.056107
284   47722698  172.60   6.033496
388   45662730  129.30   4.339397
...        ...     ...        ...
18942 35403664  109.48   3.563945
18958 31823598  119.76   3.966147
18976 32583640   95.80   3.028720
18978 32603672   98.51   3.134748
18983 33023682  141.70   4.824543
```

In this example, the correct solution was provided by all models except solar. However, most models needed a small adjustment, by adding the `nan_policy='omit'` line inside the z score calculations. As can be seen from the table, all models have a very high cosine similarity, while the lowest expectedly belongs to Solar, which is a high 90%.

Infer 9

Models\Metrics	meteor	rouge r	rouge p	rouge f	bleu	cossim	AST	levenshtein
gradient	0,4274	0,64	0,4706	0,5424	0,2177	0,8308	0	359
llama	0,4907	0,5405	0,5882	0,5634	0,2587	0,8625	0	257
openhermes	0,1187	0,25	0,2059	0,2258	0,0691	0,8293	0	279
mistral3	0,4164	0,4286	0,4412	0,4348	0,2282	0,9102	0	260
starcoder	1	1	1	1	1	1	1	0
solar	0,4714	0,5714	0,5882	0,5797	0,3159	0,9491	0	324
mistral2	0,1459	0,4074	0,3235	0,3607	0,0803	0,8505	0	288
zephyr	0,4958	0,5135	0,5588	0,5352	0,3007	0,9239	0	238

User Query: **Perform a hypothesis test to determine if the mean 'K' content in the entire dataset is significantly different from 2%. Use a t-test for the hypothesis test.**

Here is the reference code:

```
import pandas as pd
import scipy.stats as stats

# Load the dataset
df = pd.read_csv('/home/fkriskov/diplomski/datasets/lucas-soil-2018-
v2/lucas-soil-2018 copy.csv')

# Perform a t-test to determine if the mean 'K' content is significantly
different from 2%
t_stat, p_value = stats.ttest_1samp(df['K'].dropna(), 2)

# Print the results
print("T-statistic:", t_stat)
print("P-value:", p_value)
```

The correct answer: T-statistic: 134.4303152369844, P-value: 0.0

This example was correctly solved by Starcoder, Mistral3, Solar and Openhermes, while other models either had execution error or filtered incorrect data for the t-test, meaning they returned an incorrect result. Here is Llama's code that used the wrong t-test from scipy, which resulted in NaN values:

```
import pandas as pd
from scipy.stats import ttest_ind

df = pd.read_csv('/home/fkriskov/diplomski/datasets/lucas-soil-2018-
v2/lucas-soil-2018 copy.csv')

k_mean, _ = df['K'].mean(), df['K'].std()
t_stat, p_val = ttest_ind(df['K'], [2], equal_var=False)

print(t_stat, p_val)
```

Infer 10

Models\Metrics	meteor	rouge r	rouge p	rouge f	bleu	cossim	AST	levenshtein
gradient	0,4274	0,64	0,4706	0,5424	0,2177	0,8308	0	359
llama	0,4907	0,5405	0,5882	0,5634	0,2587	0,8625	0	257
openhermes	0,1187	0,25	0,2059	0,2258	0,0691	0,8293	0	279
mistral3	0,4164	0,4286	0,4412	0,4348	0,2282	0,9102	0	260
starcoder	1	1	1	1	1	1	1	0
solar	0,4714	0,5714	0,5882	0,5797	0,3159	0,9491	0	324
mistral2	0,1459	0,4074	0,3235	0,3607	0,0803	0,8505	0	288
zephyr	0,4958	0,5135	0,5588	0,5352	0,3007	0,9239	0	238

User Query: **Calculate the p-value for the correlation between 'P' and 'K'. Determine if the correlation is statistically significant.**

Here is the reference code:

```
import pandas as pd
import scipy.stats as stats

df = pd.read_csv('/home/fkriskov/diplomski/datasets/lucas-soil-2018-
v2/lucas-soil-2018 copy.csv')

correlation_coefficient, p_value = stats.pearsonr(df['P'], df['K'])

print(p_value)
```

The correct answer:

Correlation coefficient: 0.2226117821588154

P-value: 8.231555984710444e-212

The final example the models were tested on, was correctly solved by all models except solar. Llama and Mistral3 models had the exact correct solution. Solar encountered a KeyError. The common problem with this example was that the models were asked to calculate the p-value, but often returned the correlation coefficient, which was correct, but not what was asked for.

4.4 Results conclusion

To summarize, the models often provided solutions that were very close to the correct reference code, typically fixable by someone knowledgeable in programming. However, this paper's focus is on users unfamiliar with programming. Based solely on correct results, Starcoder emerges as the best model among the eight evaluated. As detailed in subsequent sections of this chapter, Starcoder not only solved all examples but also achieved the highest metric scores most frequently. In some instances, Starcoder even produced near-perfect solutions, while other models struggled to generate any viable solution. It's worth noting that achieving these results sometimes required multiple reruns of the models until a correct solution was obtained. This process can be time-consuming and problematic if there's no reliable way to verify the correctness of the generated results. Additionally, between these reruns, some models exhibited significant differences in the content they generated without any changes to settings (like temperature) or instructions. This variability adds a layer of complexity and uncertainty to the evaluation process.

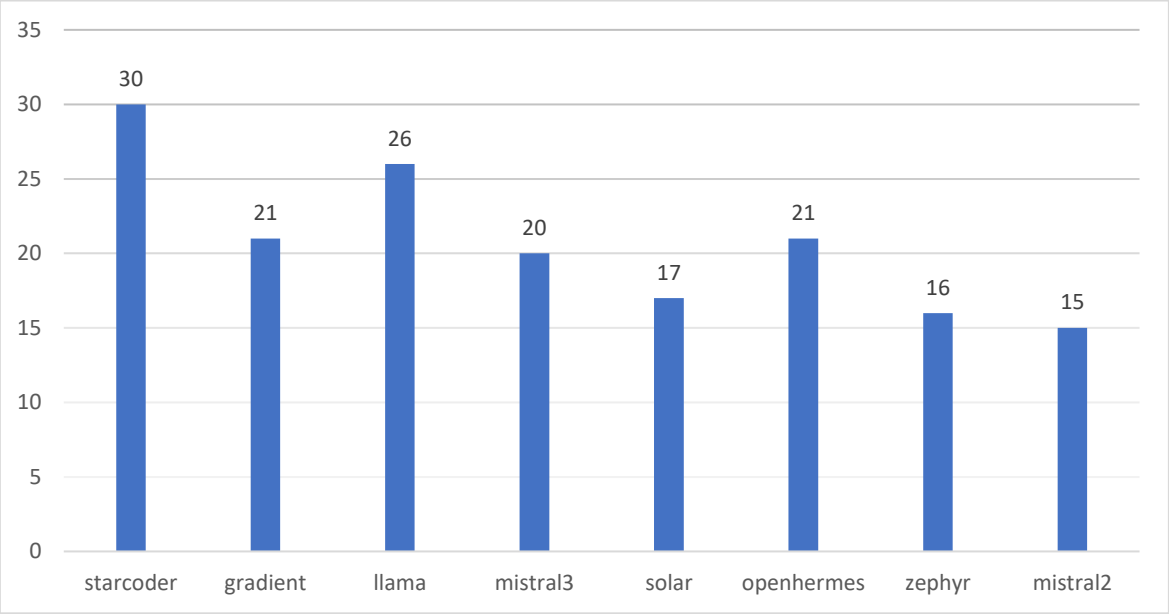
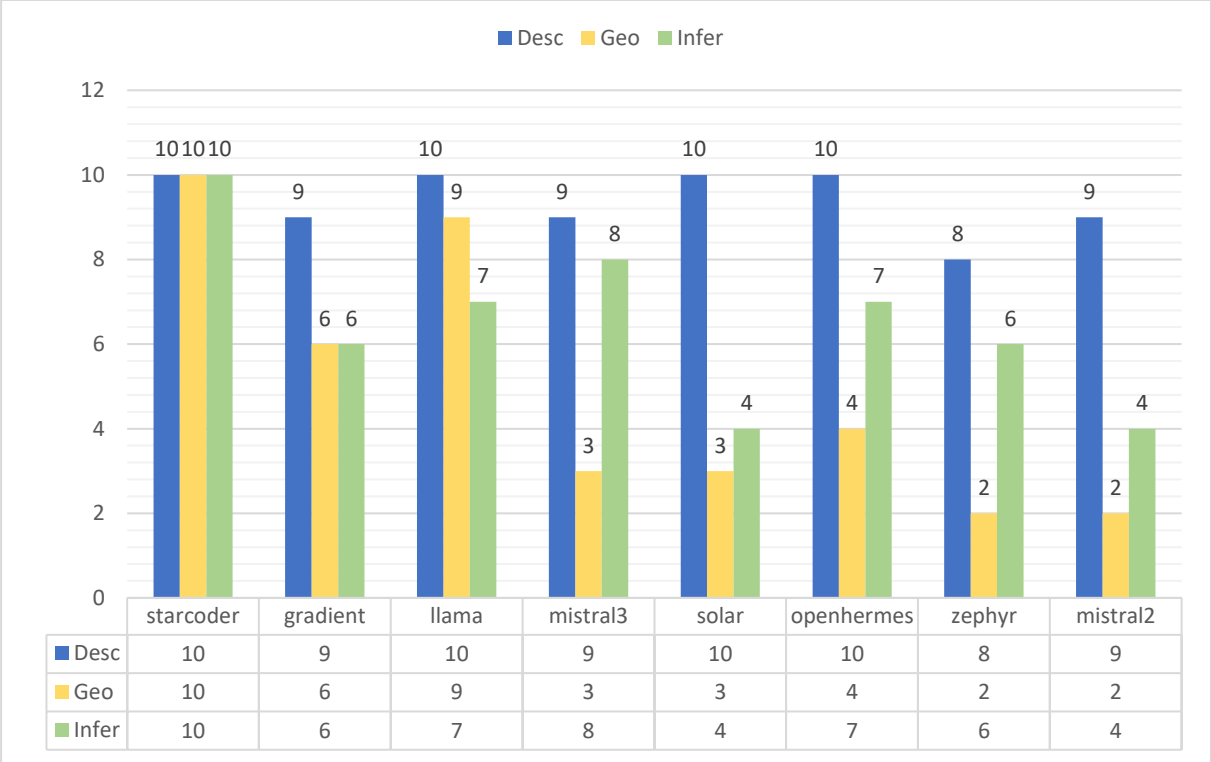


Figure 48, graph of total number of correct results

This chart summarizes the results table from the beginning of section 4.3. Starcoder stands out as the most successful model, correctly solving all test examples. Notably, llama, openhermes, gradient and mistral3 also demonstrate impressive performance, especially considering that all four have half the number of parameters compared to Starcoder. This next graph shows the total number of correct results by example categories. As explained in section 4.1, example groups are divided into descriptive statistics (Desc), inferential statistics (Infer) and geo-spatial tasks (Geo). Looking at that graph, it can be concluded that all models are proficient in solving Desc tasks as those are the simplest to implement, as well as solvable in multiple ways. The worst was zephyr, which incorrectly solved only 2 examples. The Geo and Infer questions had a much lower solving success rate, as those examples required multi-step solution. For those examples, models usually hallucinated and were generating unnecessary lines of code, with little to no value for the final solution. Llama was especially successful, alongside Starcoder, with the geo-

spatial examples, while being on the lower end with the questions in inferential statistics. Another interesting observation from these graphs is the fact that openhermes proved to be better or equal in all categories than mistra3, a model it was based on. But, on the other hand, gradient was based on meta-llama, and was worse than llama on all examples.



	starcoder	gradient	llama	mistral3	solar	openhermes	zephyr	mistral2
Desc	10	9	10	9	10	10	8	9
Geo	10	6	9	3	3	4	2	2
Infer	10	6	7	8	4	7	6	4

Figure 49, graph of total number of correct results by example category

The following graph compares two metrics across different models for all examples: the number of times a model achieved the highest score in any metric ("Top model") and the number of times a model's score was within 5% of the highest score ("Over 95%"). For instance, if Starcoder had the highest cosine similarity score of 0.9313 for a specific example like Geo 7, it would be counted in the "Top model" category. Conversely, if models such as Gradient or Llama had scores of 0.9171 and 0.9231 respectively for the same metric and example, these would not count towards the "Top model" category but would be included in the "Over 95%" category since their scores are within 5% of the top score of 0.9313.

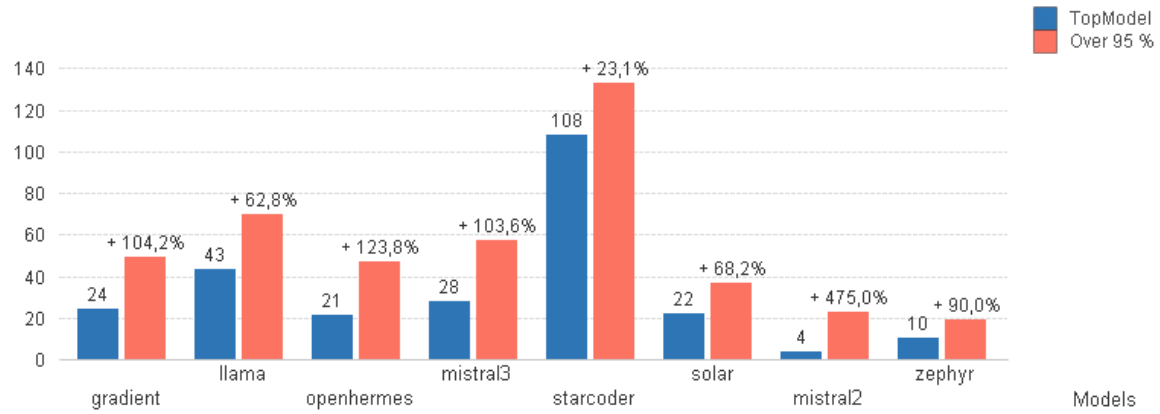


Figure 50, Comparison between the count of Top Model and Over 95% of Top Model

The percentages displayed on the red columns represent the increase in the number of instances where a model's score was within 5% of the top score ("Over 95%") compared to the instances where the model achieved the highest score ("Top Model"). This graph highlights the models that achieved the best performance most frequently, with Starcoder leading, followed by Llama, Mistral3, and Gradient. These results align with previous graphs showing correct results, except for OpenHermes, which, although rarely the top model, often had correct solutions. The graph emphasizes this scenario where a model rarely had the best overall metric but frequently had very good results.

The most significant differences are seen in Mistral2 (+475.0%), Mistral3 (+103.6%), and OpenHermes (+123.8%), while Starcoder shows the smallest difference (+23.1%). This small difference for Starcoder is expected since it has the most top results. In contrast, models with high differences had good scores often but were frequently outperformed by the top models.

Displaying results this way is crucial as it provides a clearer picture of how often models performed well in comparison to others. This indicates that the “Top Model” metric might not be the best indicator of a model's overall quality. Instead, considering the “Over 95%” category offers a more comprehensive comparison of model performance.

The following three graphs represent the "Top Model" count grouped by three example categories. These comparisons provide insight into each model’s performance across different example groups, highlighting their strengths and weaknesses.

From the graphs, the Gradient model excels in descriptive statistics, ranking in the top 3 alongside Llama and Starcoder, but performs significantly worse in geo-spatial and inferential examples. The most consistent models are Starcoder, Llama, and Solar, maintaining a similar ratio of "Top Model" counts across all categories. In contrast, OpenHermes and Mistral3, like Gradient, perform well in statistical tasks but show a notable decline in geo-spatial examples. It is also impressive that the Mistral3 model had almost the same number of “Top Models” as Starcoder in inferential statistics, which was to be expected since Mistral3 solved eight out of ten Infer examples.

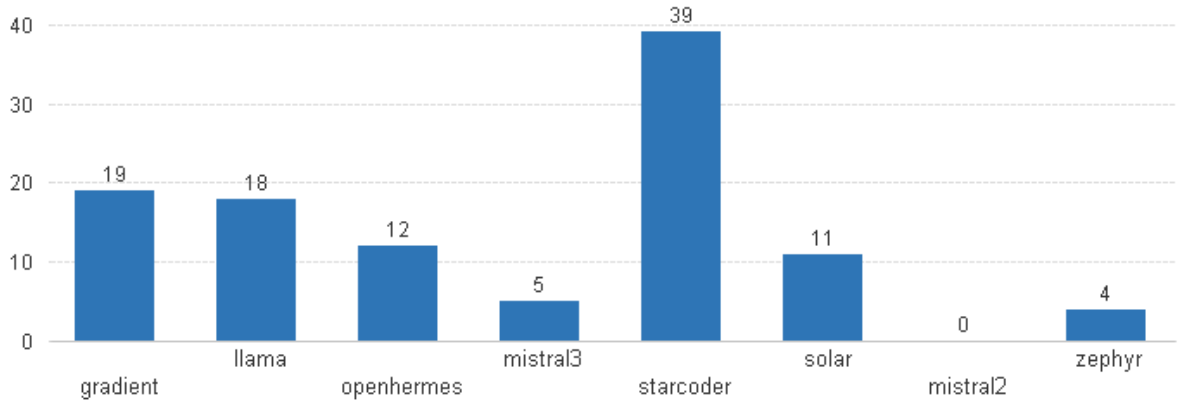


Figure 51, count of top performers in Desc examples

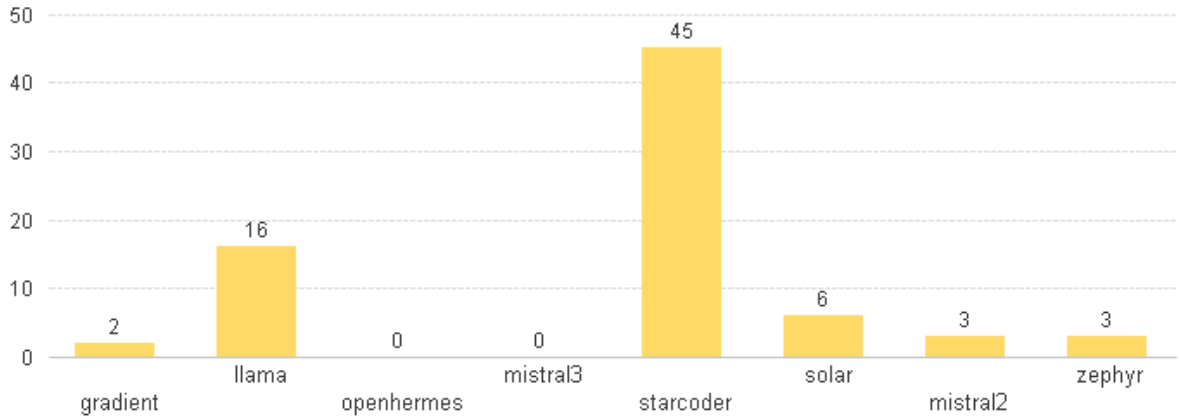


Figure 52, count of top performers in Geo examples

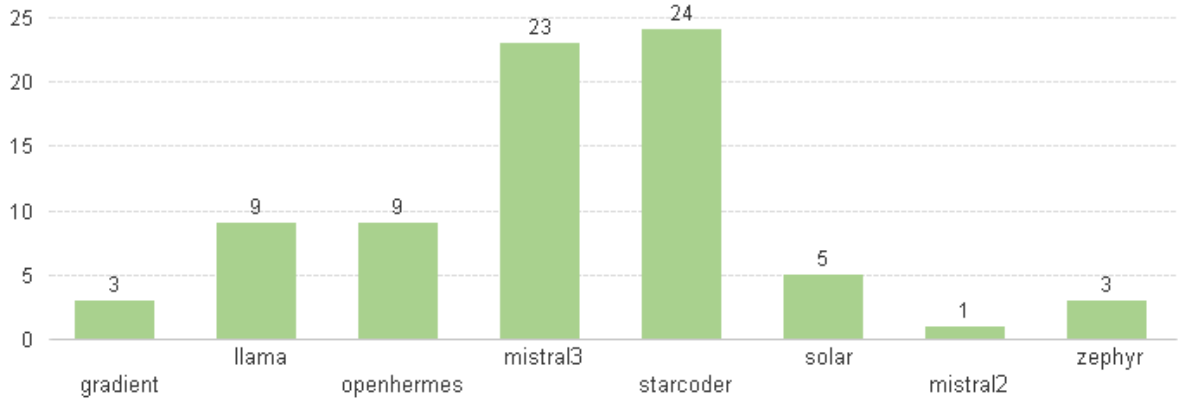


Figure 53, count of top performers in Infer examples

Another way to represent the performance and quality of the tested models is by analysing their metric scores. The following figures display graphs showing the median, highest, and lowest metric scores, as well as the average of the highest and lowest scores for each model, categorized by example type. The best and worst metric scores illustrate the range within which the models' scores fall. By comparing the median and the average, we can gauge which models generally performed well and which did not. The difference between the median and the average, indicated by the numbers above the models' names, reveals how many scores were in the top quantile. A higher score suggests that more metric scores were among the best, indicating better overall performance. The metrics that were chosen for these comparisons are METEOR and cosine similarity, as these give the best representation of how good the solutions were. METEOR is considered particularly accurate since it takes into account both precision and recall of the reference code and the generated code.

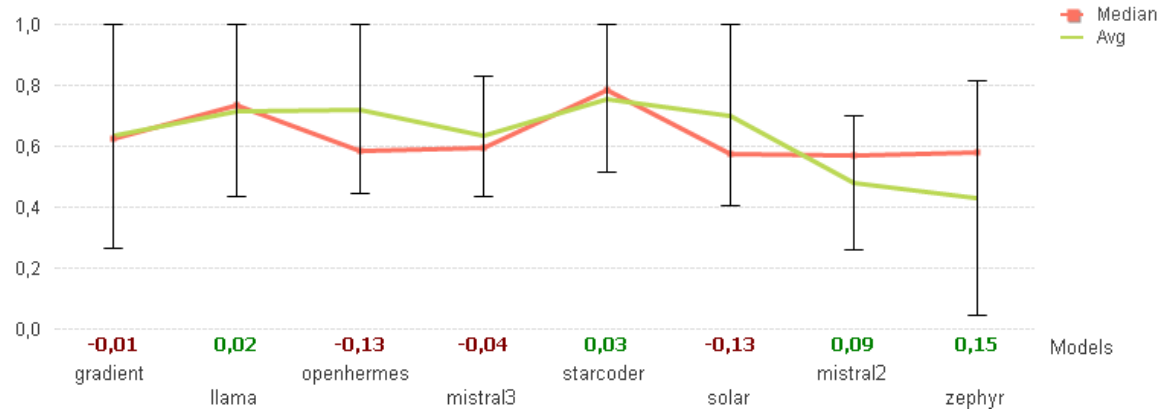


Figure 54, Difference in Median vs Average for METEOR with Desc examples

As seen from this first graph, all models except openhermes and solar had their median values above the average value, or very close to it, and the best range of METEOR scores would belong to starcoder, llama, solar and openhermes. Despite having all correct solutions for all the descriptive statistics tasks, the low scores from openhermes and solar in this category is due to there being multiple correct solutions in this category. For example, mistral2 had mostly the same solutions as the reference code, but its highest metrics score is still low, but it has the second highest median to average distance of 0.09. The highest median to average distance belongs to Zephyr, this could signify that the minimum is an outlier.

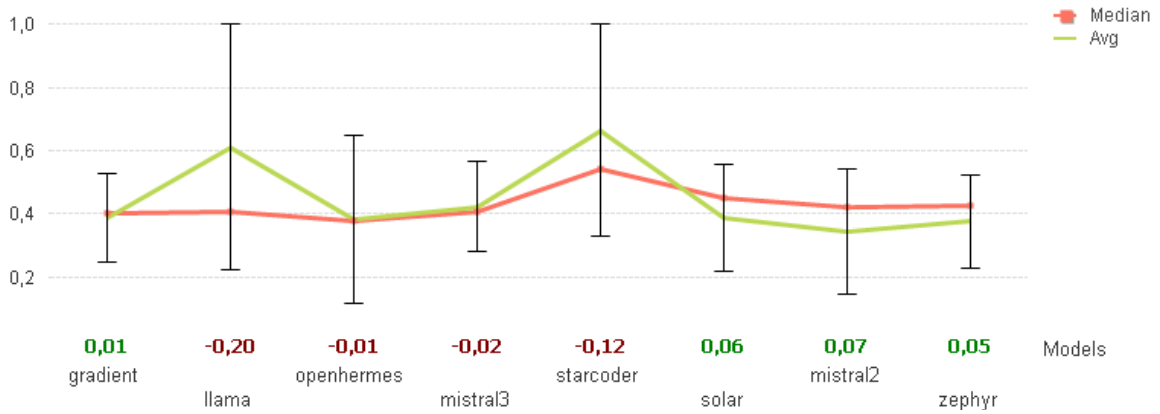


Figure 55, Difference in Median vs Average for METEOR with Geo examples

This second graph focuses on the same metric, specifically within the Geo category, revealing that the models generally produced lower scores compared to the previous category. The highest scores are notably modest, with Starcoder and Llama standing out as exceptions. However, the difference between the highest and lowest scores is smaller across models, indicating greater consistency in performance. Most models show medians that are lower than their averages, with higher differences among Solar, Mistral2 and Zephyr. Despite Solar achieving good scores overall, it correctly solved only three out of ten geo-spatial examples. This suggests that while Solar's code was generally accurate, it occasionally failed to generate the correct answer. As discussed in section 4.3, many generated codes exhibited issues that might not trouble a programmer but could present challenges for non-programmers, as these issues are not easily fixable without programming knowledge. It is also worth noting that even though Llama generated an exact code in one or two examples, its median is significantly lower than the average meaning that generally Llama performed poorly on geo-spatial examples.

Figure 56, Difference in Median vs Average for METEOR with Infer examples

Finally, this graph displays METEOR scores for inferential examples. Similar to geo-spatial tasks, all scores are generally lower compared to Starcoder, but most are around or above their respective averages, except for Llama, Openhermes and Mistral2.Mistral3, Llama and Openhermes achieved the most correct solutions, with eight and seven out of ten, following closely behind Starcoder, which achieved all ten correct solutions. Mistral3 and Gradient stand

out as the most consistent performers, maintaining scores close to their averages. However, Solar shows the largest gap between its median and average scores.

The following three graphs depict model comparisons across categories using cosine similarity. Across all examples, cosine similarity tends to yield higher scores compared to other metrics, likely due to its simplicity. This is because even when two source codes differ in solving an example, they may share common sequences such as for loops, data loading, result printing, and plot initialization, which contribute to a high cosine similarity score.

The highest and lowest values for all models are generally high, even in cases where the median is below the average, such as with Starcoder in the Geo category. Despite this, the medians themselves are quite high, indicating strong performance overall. Therefore, a lower median relative to the average does not necessarily indicate poorer performance in these instances. It is also interesting to note that all models achieved positive median-average distance on the Infer examples, even though not all models solved all inferential examples successfully. This just means that all generated solutions were really close to the correct answer, but incorrect, and probably with a small mistake, that a user couldn't fix himself.

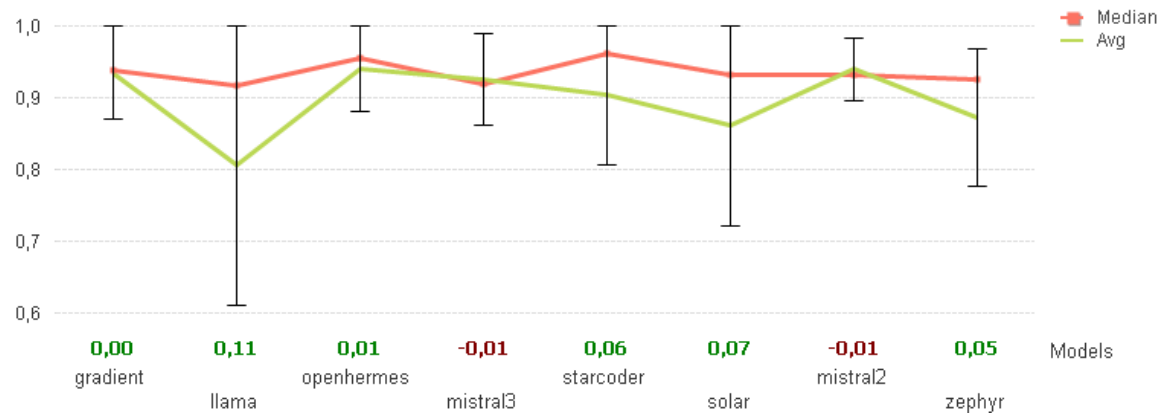


Figure 57, Difference in Median vs Average for COSSIM with Desc examples

Figure 58, Difference in Median vs Average for COSSIM with Geo examples

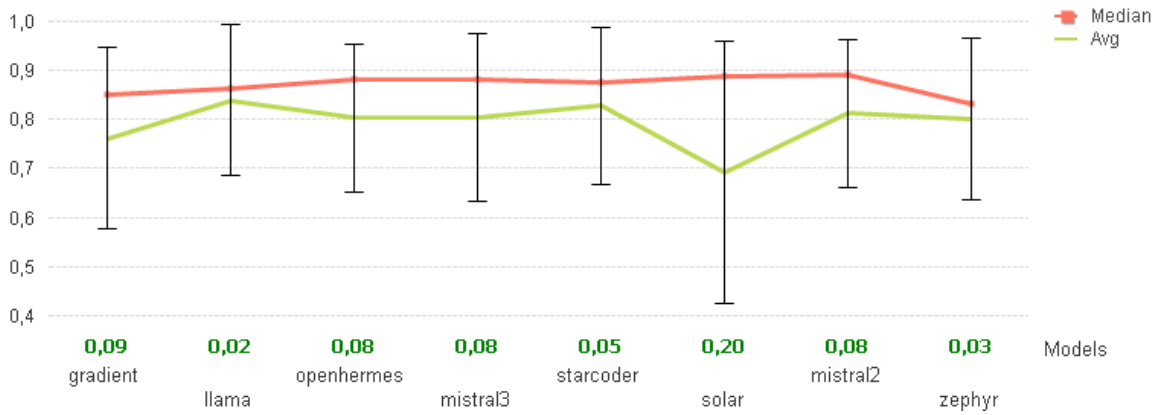


Figure 59, Difference in Median vs Average for COSSIM with Infer examples

5. Discussion

5.1 Theoretical implications

The importance of this paper lies in the fact that the system initially knows nothing about the loaded csv file and is nevertheless able to extract valuable information that the user requests. This implies that the loaded file isn't important, and that the system isn't dependent on it, which signifies that any other file could be inputted in the system, and it would work the same way. This would mean that an assistant like this can help people answer questions about a text file, in a form of a pdf, write its summaries, answer specific questions, find citations, look for pictures that contain captions or hidden alternative texts. It could probably be helpful even with video files, or image files, helping with subtitles, or even visual editing. This scalability implies that the system could adapt to handle large datasets and more complex queries without changing its core processes, which makes it a robust tool with large number of practical applications.

5.2 Practical implications

A system like the one described in this paper could significantly improve the workflow of expert data analysts by streamlining many of the tedious tasks associated with data processing. The system's ability to work with different file types without prior knowledge of the data could improve productivity and efficiency by allowing the user to only think about the information that is needed from the data, and not about the way the data can be extracted. This is especially beneficial when routine data processing tasks are needed to even start the information extraction and analysis. An example of this would be transferring one data file format into some other programme that specializes in data analysis. A system like this could take in the data and transform it and output it as a file that other applications could instantly use allowing the user to start analysing the data faster. Other workflow improvements would include the enhancement of data exploration, improved accuracy and versatility, and time and cost saving.

5.3 Conclusions

This research paper discussed and presented how to implement a language assistant for statistical analysis using large language models. The goal was to implement a system that would help people who are familiar with statistics, but aren't familiar with programming, to generate code, and execute it based on the user's question regarding a preloaded csv file. Based on [1], an autonomous agent such as this one should achieve five distinct objectives: self-generating, self-organizing, self-verifying, self-executing, and self-growing. This papers' assistant implemented self-generating and self-executing. This was achieved by testing different large language models capabilities as the brains of the system. Eight different models were tested and evaluated based on the correctness of their results as well as the quality of their results compared to a reference code. There were 30 different examples that the models were tested on, which were divided into three categories: descriptive statistics, inferential statistics and geo-spatial tasks. The most successful model was Starcoder (bigcode/starcoder2-15b-instruct-v0.1) from Hugging Face (https://huggingface.co/bigcode/starcoder2-15b-instruct-v0.1). A big part of this model's success lies in the fact that Starcoder was the model with the highest number of parameters, somewhere around 15 billion, which is significantly larger than other models with half as much. This is one of this paper's conclusions; bigger models produce better results. There are also some other aspects to this, but the number of parameters plays a huge role in the model's performance. The reason why bigger models weren't tested to further analyse this assumption is because of resource limitations. More parameters means bigger models, and bigger models require more virtual memory on a graphics card to properly function or function at all. Another conclusion is that the models are very susceptible to the way user queries and instructions are formed, and their output is heavily influenced by even a small change in the provided context. With that in mind, the models are very prone to hallucinations, some more than others. At times, models listen to every instruction and rule, and use all the available information to generate the most accurate results, while, on the other hand, they can as easily generate a solution with a syntax error, empty solution graph or a code that is against the provided instructions. This also means that, providing the same set of instructions is given and no changes are made to the initial parameters, the models generate very different results between two executions. Finally, the test examples show that the models are more efficient at solving simpler problems, rather than the ones that require step-by-step solutions. This was somewhat fixed with the implementation of a step generation process, that is followed by a separate code generation process. This, in theory, should partly solve the problem of complex queries and allow models to more accurately generate correct and quality results even for more difficult tasks. The reason this only partly solves the problem is because, even though the steps are generated, as mentioned above, the models won't necessarily follow them. Another important observation, as seen in [1], is that it is more beneficial to give the language models a full context when translating steps into code, rather than giving them one step to implement at a time. This improves the coherence between steps, especially is the steps require an implementation of a function, a full context is needed so that the function arguments, and function's return type is correctly defined. This should be the case for any upgrades to the system and any additional processes - providing the full context, generated steps and the generated code.

5.4 Limitations and future research

One of the primary limitations encountered in this study was the hardware capacity, specifically the amount of VRAM available on the graphics cards. This significantly constrained the size of the models that could be utilized. Since the models were executed using CUDA, the VRAM capacity directly influenced the number of parameters that could be loaded simultaneously. Consequently, this restricted the selection of models, preventing some larger models, such as “meta-llama/Meta-Llama-3-70B-Instruct”, from being loaded into the VRAM due to their extensive parameter size.

Another critical aspect influenced by these hardware restrictions was the ability to fine-tune the models. Fine-tuning could significantly improve the model's understanding of specific rules and instructions, leading to more accurate results. However, due to VRAM limitations, fine-tuning was not feasible for larger models. This often resulted in models disregarding some instructions or restrictions, opting for simpler but incorrect solutions. Similarly, building and training a model from scratch were also hindered by the hardware constraints.

As discussed in 4.3, in the Geo 2 example, some models lacked the capacity to adhere to constraints and instructions, resulting in solutions that conflicted with the set restrictions. A potential improvement for future work could involve guiding the model's solution rather than imposing restrictions for the models to navigate around. This approach might yield better results by encouraging the models to inherently think of a certain solution without considering other possibilities, provided they receive appropriate guidance.

Another thing that could be improved is the way the double check works. Currently, it receives the complete code and the complete instructions and tries to identify all instances where the code does not follow the instructions. Having the model check a rule at a time could prove to be better in terms of finding conflicts in the code, on the other hand, instead of checking the rules, checking lines of code one at a time could produce similar effects. This would probably function significantly better with the help of fine-tuning, which would require a dataset, that doesn’t necessarily have to be large. This dataset would have to contain a code, a set of rules, and identified lines that need to be changed for the code to work. Additionally, since rules don’t change a lot within the same group of questions, this could prove to be even more effective.

An effective method that was seen both in this paper and in [1] is the implementation of a step generation process. The steps generation could be improved upon by implementing a clever agent or an LLM that specializes in solving problems in steps. If such model is fine-tuned or instructed to have in mind a problem solution that can be executed in code, that would significantly improve the steps generation process and would yield better results. Additionally, such agent could be trained or encouraged to use some pre-determined steps that the code generation model can easily implement. These pre-determined steps could be some simple operations that would include preparation of graph drawing, immediate base map drawing (Europe shapefile) or data filtering preparation. The code generation model would treat these operations like function calls, by either calling the pre-implemented function that provides the requested functionality or by being fine-tuned to implement such a functionality once requested to implement a specific pre-defined step.

With the mention of other models taking over certain steps in the system, another improvement that could be made is a model that would be in charge of coming up with instructions and rules for the step generation or code generation processes. This could help expand the assistant and enable it to process some other forms of input files and answer different types of questions that are specific to the type of file provided.

6. References

[1] Li Z, Ning H. Autonomous GIS: the next-generation AI-powered GIS. International Journal of Digital Earth. 2023;16(2):4668-4686. doi: 10.1080/17538947.2023.2278895

[2] R. Merritt, "What Is Retrieval-Augmented Generation, aka RAG?," NVIDIA, 15 November 2023. [Online]. Available: https://blogs.nvidia.com/blog/what-is-retrieval-augmented-generation/.

[3] Zhao, P., Zhang, H., Yu, Q., Wang, Z., Geng, Y., Fu, F., ... & Cui, B. (2024). Retrieval-augmented generation for ai-generated content: A survey. *arXiv preprint arXiv:2402.19473*. https://doi.org/10.48550/arXiv.2402.19473

[4] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., ... & Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, *33*, 9459-9474. https://doi.org/10.48550/arXiv.2005.11401

[5] A. A. Awan, "An Introduction to Pandas AI," Datacamp, January 2023. [Online]. Available: https://www.datacamp.com/blog/an-introduction-to-pandas-ai.

[6] P. Krampah, "Chat With Your CSV File With PandasAI," Medium, 26 September 2023. [Online]. Available: https://medium.com/aimonks/chat-with-your-csv-file-with-pandasai-22232a13c7b7.

[7] Sudarshan, "Introduction to PandasAI Part 1," Medium, 23 March 2024. [Online]. Available: https://medium.com/@sudarshan.gamakaai/introduction-to-pandasai-part-1-a7bc8721a3f3.

[8] S. Dey, "PandasAI: Simplifying Data Analysis through Generative AI," Medium, 10 March 2024. [Online]. Available: https://medium.com/@soumava.dey.aig/pandasai-simplifying-data-analysis-through-generative-ai-980b73a410ff.

[9] Wang, L., Xu, W., Lan, Y., Hu, Z., Lan, Y., Lee, R. K. W., & Lim, E. P. (2023). Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. *arXiv preprint arXiv:2305.04091*. https://doi.org/10.48550/arXiv.2305.04091

[10] LangChain, "Plan-and-Execute Agents," LangChain, 13 February 2024. [Online]. Available: https://blog.langchain.dev/planning-agents/.

[11] LangChain, “Plan and execute,” LangChain, 2024. [Online]. Available: https://js.langchain.com/v0.1/docs/modules/agents/agent_types/plan_and_execute/.

[12] LangChain, “langgraph examples llm-compiler,” Github, 2024. [Online]. Available: https://github.com/langchain-ai/langgraph/blob/main/examples/llm-compiler/LLMCompiler.ipynb.

[13] LangChain, “langgraph examples plan-and-execute,” Github, 2024. [Online]. Available: https://github.com/langchain-ai/langgraph/blob/main/examples/plan-and-execute/plan-and-execute.ipynb.

[14] LangChain, “langgraph examples rewoo,” Github, 2024. [Online]. Available: https://github.com/langchain-ai/langgraph/blob/main/examples/rewoo/rewoo.ipynb.

[15] Liu, Z., Qiao, A., Neiswanger, W., Wang, H., Tan, B., Tao, T., ... & Xing, E. P. (2023). Llm360: Towards fully transparent open-source llms. *arXiv preprint arXiv:2312.06550*. https://doi.org/10.48550/arXiv.2312.06550

[16] Acikgoz, E. C., İnce, O. B., Bench, R., Boz, A. A., Kesen, İ., Erdem, A., & Erdem, E. (2024). Hippocrates: An Open-Source Framework for Advancing Large Language Models in Healthcare. *arXiv preprint arXiv:2404.16621*. https://doi.org/10.48550/arXiv.2404.16621

[17] Labrak, Y., Bazoge, A., Morin, E., Gourraud, P. A., Rouvier, M., & Dufour, R. (2024). Biomistral: A collection of open-source pretrained large language models for medical domains. *arXiv preprint arXiv:2402.10373*. https://doi.org/10.48550/arXiv.2402.10373

[18] Chen, D., Huang, Y., Li, X., Li, Y., Liu, Y., Pan, H., ... & Han, K. (2024). Orion-14b: Open-source multilingual large language models. *arXiv preprint arXiv:2401.12246*. https://doi.org/10.48550/arXiv.2401.12246

[19] Di, P., Li, J., Yu, H., Jiang, W., Cai, W., Cao, Y., ... & Zhu, X. (2024, April). Codefuse-13b: A pretrained multi-lingual code large language model. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice* (pp. 418-429). https://doi.org/10.1145/3639477.3639719

[20] Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., ... & Sifre, L. (2022). Training compute-optimal large language models. arXiv 2022. *arXiv preprint arXiv:2203.15556*, *10*. https://doi.org/10.48550/arXiv.2203.15556

[21] Zan, D., Chen, B., Zhang, F., Lu, D., Wu, B., Guan, B., ... & Lou, J. G. (2022). Large language models meet nl2code: A survey. *arXiv preprint arXiv:2212.09420*. https://doi.org/10.48550/arXiv.2212.09420

[22] Zhang, P., Zeng, G., Wang, T., & Lu, W. (2024). Tinyllama: An open-source small language model. *arXiv preprint arXiv:2401.02385*. https://doi.org/10.48550/arXiv.2401.02385

[23] Li, D., Shao, R., Xie, A., Sheng, Y., Zheng, L., Gonzalez, J., ... & Zhang, H. (2023). How Long Can Context Length of Open-Source LLMs truly Promise?. In *NeurIPS*

2023 Workshop on Instruction Tuning and Instruction Following. https://openreview.net/forum?id=LywifFNXV5

[24] Z. Keita, "Llama.cpp Tutorial: A Complete Guide to Efficient LLM Inference and Implementation," Datacamp, November 2023. [Online]. Available: https://www.datacamp.com/tutorial/llama-cpp-tutorial.

[25] Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M. A., Lacroix, T., ... & Lample, G. (2023). Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*. https://doi.org/10.48550/arXiv.2302.13971

[26] Naveed, H., Khan, A. U., Qiu, S., Saqib, M., Anwar, S., Usman, M., ... & Mian, A. (2023). A comprehensive overview of large language models. *arXiv preprint arXiv:2307.06435*. https://doi.org/10.48550/arXiv.2307.06435

[27] Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., Casas, D. D. L., ... & Sayed, W. E. (2023). Mistral 7B. *arXiv preprint arXiv:2310.06825*. https://doi.org/10.48550/arXiv.2310.06825

[28] Tunstall, L., Beeching, E., Lambert, N., Rajani, N., Rasul, K., Belkada, Y., ... & Wolf, T. (2023). Zephyr: Direct distillation of lm alignment. *arXiv preprint arXiv:2310.16944*. https://doi.org/10.48550/arXiv.2310.16944

[29] teknium, "huggingface.co/teknium/OpenHermes-2.5-Mistral-7B," Huggingface, 2023. [Online]. Available: https://huggingface.co/teknium/OpenHermes-2.5-Mistral-7B.

[30] Kim, D., Park, C., Kim, S., Lee, W., Song, W., Kim, Y., ... & Kim, J. Solar 10.7 b: Scaling large language models with simple yet effective depth up-scaling. arXiv 2023. *arXiv preprint arXiv:2312.15166*. https://doi.org/10.48550/arXiv.2312.15166

[31] gradientai, "huggingface.co/gradientai/Llama-3-8B-Instruct-Gradient-1048k," Huggingface, 2024. [Online]. Available: https://huggingface.co/gradientai/Llama-3-8B-Instruct-Gradient-1048k.

[32] Lozhkov, A., Li, R., Allal, L. B., Cassano, F., Lamy-Poirier, J., Tazi, N., ... & de Vries, H. (2024). Starcoder 2 and the stack v2: The next generation. *arXiv preprint arXiv:2402.19173*. https://doi.org/10.48550/arXiv.2402.19173

[33] J. L. Espejel, "Basic understanding of Abstract Syntax Tree (AST)," Medium, 2 January 2023. [Online]. Available: https://medium.com/@jessica_lopez/basic-understanding-of-abstract-syntax-tree-ast-d40ff911c3bf.

Implementation of a language assistant for statistical analysis using large language models

Abstract

This thesis investigates the need and process of creating a language assistant that provides answers to user queries related to CSV files. The implementation of the assistant relies on the use of available open source large language models adapted to the user's requirements. User queries are transformed from natural to programming language in order to make it possible to perform statistical calculations on the entered data and provide the user with answers in natural language. The language assistant's performance will be evaluated using appropriate metrics in order to assess its effectiveness and reliability in providing answers to the questions posed.

Keywords – Natural Language Processing; chatbots; code generation; transformers; Statistical Analysis; Spatial Analysis

Paper Type – Research paper

Implementacija jezičnog asistenta za statističke analize korištenjem velikih jezičnih modela

Sažetak

Ovaj diplomski rad istražuje potrebu i proces izrade jezičnog asistenta koji pruža odgovore na upite korisnika vezane uz CSV datoteke. Implementacija asistenta se oslanja na korištenje dostupnih velikih jezičnih modela otvorenog koda, prilagođenih zahtjevima korisnika. Korisnički upiti se transformiraju iz prirodnog u programski jezik kako bi se omogućilo izvođenje statističkih izračuna nad unesenim podacima te korisniku pružili odgovori u prirodnom jeziku. Odgovarajućim metrikama provest će se evaluacija performansi jezičnog asistenta kako bi se procijenila njegova učinkovitost i pouzdanost u pružanju odgovora na postavljena pitanja.

Ključne riječi – Obrada Prirodnog Jezika; chatbotovi; generiranje koda; transformeri; Statistička analiza; Prostorna analiza

Vrsta rada – istraživački rad