Karl Oskar Magnus Holm

LLMs - The Death of GIS Analysis?

An Investigation into Using Large Language Models for GIS Data Analysis

Master Thesis in Computer Science and Geomatics, June 2024
Supervisor at NTNU: Hongchao Fan
External supervisors from Norkart: Alexander Salveson Nossum, Arild Nomeland, and Rune Aasgaard

Department of Geomatics
Faculty of Engineering
Norwegian University of Science and Technology

Abstract

List of Figures

List of Tables

1. Introduction

The introductory chapter will explain the motivation behind the thesis, as well as its goals and the research questions it will attempt to answer. Section 8.1 will list the main contributions of the thesis, and section 1.5 will give a high-level overview over the thesis.

1.1. Background and Motivation

The release of OpenAI's ChatGPT in November, 2023 generated a hype within the general population and chat-based systems are flourishing. ChatGPT — or rather the underlying GPT-3 and GPT-4 models — is an example of how modern Large Language Models (LLMs) can provide a natural language interface between human and machine. Furthermore, significant advancements have been made within code generation, which essentially allows the LLM to execute computational tasks that can be defined within a snippet of code. Paired with the LLM's ability to often correctly interpret the user's intent regardless of the preciseness of their problem formulation, even individuals with no prior programming experience can carry out computational tasks that require the execution of code.

ref

GIS analysis has traditionally been reserved for GIS *experts*. GIS professional are commonly required to know their way around one or more Geographic Information Systems, in addition to being proficient in programming languages suitable to data science task, such as Python or R. Extensive domain knowledge is often also necessary when tackling GIS tasks, like knowing which data to use for a particular tasks and where to get them. All of these points — and more — are barriers to entry for people that wish to make use of powerful GIS tools for their particular purposes, but lack the technical know-how required to use them correctly. This challenge serves as the overall motivation behind this master's thesis, which will mitigate these issues by utilizing the vast background knowledge and code generation abilities of modern LLMs.

1.2. Goals and Research Questions

Deriving from the motivation described in the section above, the overarching goal of this master's thesis becomes to investigate the possibilities of utilizing LLMs to have a natural language interface with a system that is capable of solving GIS-related tasks. The hypothesis is that modern LLMs are embedded with an understanding of common GIS workflows, and that their code generation abilities are of such a level that they can accomplish tasks in an autonomous manner.

Based on this overarching goal, three research questions have been constructed and are listed below:

1. Can an LLM-based system perform common GIS tasks?
2. What are core challenges when developing larger systems that rely on Large Language Models to control its logic flow?
3. What are state-of-the-art methods of creating autonomous LLM-based agents?

1.3. Research Method

Prior to this master's thesis, a specialization project on the same topic was conducted. The specialization project — as detailed in (Holm, 2023) — was of a theoretical character predominantly served as a literature study leading up to the master's thesis. The research questions listed in the above section, however, call for a different and more practical approach. Specifically, RQ1 and RQ2 can only be addressed by drawing on insights gained from the attempt to develop such an LLM-based GIS system. Therefore, this master's thesis will revolve around a "proof of concept" and evaluating the usefulness of the system, as well as lessons learned from the development process.

RQ3, on the other hand, will be a continuation of the theoretical work done in the specialization process, and will serve as a foundation for the "proof of concept" development process.

1.4. Contributions

1.5. Thesis Structure

2. Background Theory

Chapter 2 will lay a theoretical basis for the work done in this master thesis, providing the user with the required understanding in order to understand the contributions of the work. Section 2.1 will explain the theoretical basis of the component which most modern Large Language Models (LLMs) are based upon — namely the Transformer — and the attention mechanism within it. The section will also touch upon a new approach to language modelling called *selective state space modelling*, which has yielded very promising results for small LLMs.

Parts of the Background chapter is reused material from the specialization project (Holm, 2023) preceding this master thesis. Below are the sections in question, together with a description of the extent to which, and how, the material is reused:

- *Subsection 2.1.4: Reused without modification.*
- *Subsection 2.3.1: Reused without modification.*

2.1. Approaches to Language Modelling

Subsection 2.1.4 will explain the theoretical basis behind most modern LLMs, which are based upon the attention mechanism built into the Transformer architecture. Subsection 2.1.5 will explain modern state space-modelling approaches and why they may have a potential greater than Transformer-based models. First, however, subsection 2.1.1 will delve into earlier attempts at language modelling.

2.1.1. Early Attempts: Statistical Models and Recurrent Neural Networks

2.1.2. Statistical Models

2.1.3. Recurrent Neural Networks

2.1.4. Attention and the Transformer Architecture

Vaswani et al. (2017) managed to achieve new state-of-the-art results for machine translation tasks with their introduction of the Transformer architecture. The Transformer has later been proved effective for numerous downstream tasks, and for a variety of modalities. Titling their paper *Attention Is All You Need*, Vaswani et al. suggest that their attention-based architecture renders network architectures like Recurrent Neural

Networks (RNNs) redundant, due to its superior parallelization abilities and the shorter path between combinations of position input and output sequences, making it easier to learn long-range dependencies (Vaswani et al., 2017, p. 6).

The Transformer employs self-attention, which enables the model to draw connections between arbitrary parts of a given sequence, bypassing the long-range dependency issue commonly found with RNNs. An attention function maps a query and a set of key-value pairs to an output, calculating the compatibility between a query and a corresponding key (Vaswani et al., 2017, p. 3). Looking at Vaswani et al.'s proposed attention function (2.1), we observe that it takes the dot product between the query Q and the keys K, where Q is the token that we want to compare all the keys to. Keys similar to Q will get a higher score, i.e., be *more attended to*. These differences in attention are further emphasized by applying the softmax function. The final matrix multiplication with the values V (the initial embeddings of the input tokens) will yield a new embedding in which all individual tokens have some context from all other tokens. We improve the attention mechanism by multiplying queries, keys, and values with weight matrices that are learned through backpropagation. Self-attention is a special kind of attention in which queries, keys, and values are all the same sequence.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.1}$$

Attention blocks can be found in three places in the Transformer architecture (Vaswani et al., 2017, p. 5) (I will use machine translation from Norwegian to German as an example):

1. In the encoder block to perform self-attention on the input sequence (which is in Norwegian)
2. In the decoder block to perform self-attention on the output sequence (which is in German)
3. In the decoder block to perform cross-attention (also known as encoder-decoder attention) where each position in the decoder attends to all positions in the encoder

The Transformer represented a breakthrough in the field of Natural Language Processing (NLP), and is the fundamental building block of modern LLMs, most famous of which are the GPT's.

2.1.5. State Space Models

2.2. Function Calling LLMs

Function calling—first introduced by OpenAI (Eleti et al., 2023)—allows developers to provide function definitions to an LLM and have said LLM output a JSON object containing the name of one or more of the functions provided, as well as suitable arguments

to these. Made possible through fine-tuning models to detect when functions should be calling, function calling makes it possible to give an LLM *hooks* into the real world, and provides a more reliable way for developers to integrate LLMs into applications.

Possible use cases include using functions provide correct and up-to-date information that would otherwise require extensive training and fine-tuning. Having the LLM use function calling for information retrieval also make them more transparent, making it possible to trace a claim back to its source, something that is normally a difficult feat with LLM. Another use case might be code execution. One could imagine a rather simple function `execute_python_code(code: string) -> string` that takes Python code as a string and returns the standard output that results from executing that code. This is likely the principle behind products like OpenAI's Data Analysis mode (previously Code Interpreter), in which ChatGPT functions as a code executing agent that can generate, execute, and self-correct its own code. Similar functions could be constructed for SQL, making it possible for LLMs to work against relational databases. As Eleti et al. (2023) describes, function calling can also be used to extract structured data from text.

2.3. State-of-the-Art Large Language Models

2.3.1. The GPT Family

Generative Pre-trained Transformer (GPT) is a type of LLM that was introduced by OpenAI in 2018 (Radford and Narasimhan, 2018). Specifically designed for text generation, a GPT is essentially a stack of Transformer *decoders*. It demonstrates through its vast pre-training on unlabelled data that such unsupervised training can help a language model learn good representations, providing a significant performance boost while alleviating the dependence on supervised learning. While the original Transformer architecture as described by Vaswani et al. (2017) was intended for machine translation—thus having encoders to learn the representation of the origin language representation of a given input sequence and decoders to learn the representation in the target language and perform cross-attention between the two—the GPT is designed only to *imitate* language. This is why there are no encoders to be found in the GPT architecture, only decoders. The model employs masked multi-head attention (running the input sequence through multiple attention heads in parallel), and is restricted to only see the last k tokens—with k being the size of the context window—and tasked to predict the next one.

Training consists of two stages: unsupervised pre-training and supervised fine-tuning. The former is used to find a good initialization point, essentially teaching the model to imitate the corpora upon which it is trained. This results in a model that will ramble on uncontrollably, just trying to elaborate upon the input sequence it's given to the best of its knowledge. This will naturally produce undefined behaviour, and it is therefore necessary to fine-tune the model on target tasks in a supervised manner. Radford and Narasimhan (2018, p. 4) explain how the model can be fine-tuned directly on tasks like text classification, but how one for other tasks needs to convert structured inputs into ordered sequences because the pre-trained model was trained on contiguous sequences

of text. In the case of ChatGPT, OpenAI used Reinforcement Learning from Human Feedback (RLHF) by employing a three-step strategy: first training using a supervised policy, then using trained reward models to rank alternative completions produced by ChatGPT models, before fine-tuning the model using Proximal Policy Optimization (PPO), which is a way of training AI policies. This pipeline is then performed for several iterations until the model produces the desired behaviour (OpenAI, 2022).

2.3.2. The Gemini Family

2.3.3. The Claude Family

2.3.4. Open-Source Alternatives

3. Related Work

3.1. LLM-based Systems in Geospatial Technologies

3.2. Agent Patterns

LLM-based agents can be implemented in many ways, and researchers have developed a plethora of *agent patterns* that seek to improve upon areas where LLMs tend to be less effective. This includes patterns for retrieval of external data, as well as multi-agent patterns. Such patterns can improve the performance of LLM-based agents within any domain, and should also be considered when developing one for geospatial purposes. The following sections will shortly explain some of the patterns that were considered in the development of GeoGPT.

3.2.1. The Multi-Agent Pattern

The multi-agent pattern that takes inspiration from human collaboration in that it is made up from multiple specialized agents that work together to achieve some objective. There have been several implementations of the pattern, with certain differences. MetaGPT (Hong et al., 2023) is a LLM-based multi-agent system consisting of agents with human-level domain expertise. Using an assembly line paradigm, where the overall goal is divided into subtasks, Hong et al. showed that MetaGPT could generate more coherent solutions compared to the previous state-of-the-art multi-agent systems. At the time of release, MetaGPT set a new state-of-the-art performance on the HumanEval and MBPP benchmarks (Hong et al., 2023, p. 7), demonstrating the potential of the multi-agent pattern.

3.2.2. Patterns for Retrieval Augmented Generation

3.2.3. Patterns for Self-Reflection

3.3. LangChain

LangChain (LangChain AI, 2022) is an open-source project that provides tooling which simplifies the way developers interface with Large Language Models (LLMs). This tooling includes composable tools and integrations that can be used to build prompts for LLMs, as well as off-the-shelf chains that perform higher level tasks. Chains are Directed Acyclic Graphs (DAGs) — or sequences of runnables — that take an input and produces and output. A runnable can be a prompt template with template literals that are substituted

with values that are passed into the runnable. The output is the template with the template literals filled in. This output can then be chained into an LLM runnable calls a language model using the prompt template. The output from the LLM runnable could then be passed into an output parser, e.g. a JSON parser, that ensures that the chain outputs a JSON object. Such chains are the buildings blocks that make up LangChain.

Common use cases for LangChain are:

- Building chatbots for question answering that use semantic retrieval from document store
- Creating agents with access to external tools be leveraging function calling (see section 2.2)
- Creating code executing agents for Python, SQL, or other programming languages

In January 2024, LangChain AI rolled out a new framework called LangGraph which builds on top of the LangChain ecosystem. While the chains commonly found in LangChain are good for DAG workflow, they are not suited to creating cyclic graphs. LangGraph can be used to add cycles to LLM applications, which are important for agent-like behaviours (LangChain AI, 2024). A graph in LangGraph is a set of nodes that pass some state around, state that can be modified by each node. The nodes are connected together by edges that define what node can succeed another node. These edges can also be conditional, which routes execution to a given node based on the output from a function giving the current state. This allows for complex logic and simplifies implementation of advanced agent patterns, some of which are discussed in section 3.2.

3.4. Geospatial Databases and Data Catalogues

4. Datasets

With the interest of investigating the ability of a Large Language Model (LLM)-based system to perform geospatial analysis, relevant datasets should be accessible to said system. Section 4.1 provides a description of the datasets used in the experiments. Furthermore, it was decided to explore different access channels to this data. Section 4.2 elaborates on this.

4.1. Data Sources

A total of eighteen datasets were used in the experiments. The data was downloaded from Geofabrik's website[1]. Geofabrik — German for "geo factory" — is a company that "extract, select, and process free geodata". They have gathered data from OpenStreetMap and published them as a collection of shapefiles, dividing them into categories such as "places of worship", "points of interest", and "traffic". Data can be downloaded for different regions of the world, and for experiments conducted in this thesis, data for Norway was used. Table 4.1 lists all datasets used, along with a short description of their contents. Common for all datasets are their *fclass* attribute, which is short for *feature class*. Some datasets have additional attributes, such as the *maxspeed* attribute in the road data and the *type* attribute in the building data.

Table 4.1.: Datasets used in experiments

Dataset	**Data Type**	**Description**
Buildings	Polygon	Contains building outlines. Its *type* attribute can have values like *house*, *university*, and *restaurant*.
Land Use	Polygon	Represents areas designated to different purposes and activities. Its *fclass* attribute can have values like *forest*, *farmland*, and *residential*.
Natural	Point	Contains outlines of various objects found in nature. Its *fclass* attribute can have values like *beach*, *glacier*, and *cave_entrance*.

Continued on next page

[1] https://download.geofabrik.de/europe/norway.html

Table 4.1 continued from previous page

Dataset	Data Type	Description
Natural	Polygon	Similar to the point data equivalent.
Places of Worship	Point	Common values for *fclass* attribute: *christian*, *buddhist*, and *muslim*.
Places of Worship	Polygon	Similar to the point data equivalent.
Places	Point	Common values for *fclass* attribute: *farm*, *village*, and *island*. Repeated entries trimmed for brevity.
Places	Polygon	Similar to the point data equivalent.
Points of Interest	Point	Common values for *fclass* attribute: *tourist_info*, *bench*, and *kindergarten*.
Points of Interest	Polygon	Similar to the point data equivalent.
Railways	Lines	Common values for *fclass* attribute: *rail*, *subway*, and *tram*. Also has True/False attributes saying if a given line segment is a bridge or a tunnel.
Roads	Lines	Common values for *fclass* attribute: *rail*, *subway*, and *tram*. Has additional attributes *oneway*, *maxspeed*, *bridge*, and *tunnel*.
Traffic	Point	Common values for *fclass* attribute: *crossing*, *street_lamp*, and *parking*.
Traffic	Polygon	Common values for *fclass* attribute: *parking*, *pier*, and *dam*.
Transport	Point	Common values for *fclass* attribute: *bus_stop*, *ferry_terminal*, and *railway_station*.
Transport	Polygon	Similar to the point data equivalent.
Water	Polygon	Common values for *fclass* attribute: *water*, *wetland*, and *river_bank*.
Waterways	Lines	Common values for *fclass* attribute: *stream*, *river*, and *canal*.

4.2. Data Access

While leading LLMs are trained on increasingly large corpora, they are still only as familiar with a topic as the extent to which the training data exposes it to said topic. For instance, many LLMs are trained specifically to generate Python code, and are

therefore fed with a vast number of Python code examples during training in the hopes of improving its performance on benchmarks like . As it is unlikely that the training data is evenly distributed among many different topics, it is useful to get familiarized with a model's capabilities in the areas of interest for a particular use case. In the case of an LLM-powered GIS agent that should be capable of performing geospatial analyses, it is useful to know what data formats such an agent is most comfortable to understand and work with.

Insert Python benchmark examples here

The upcoming experiments therefore seek to benchmark model performance on three different data access methods. The datasets from section 4.1 are presented to the model in three different ways, as subsection 4.2.1 through subsection 4.2.2 elaborate upon.

4.2.1. Files

The first method of presentation is to have the files from section 4.1 remain untouched. The datasets were stored such that each dataset has its own folder. This is because some of the file types used require multiple files in order to correctly store the data—for instance, the shapefile format, which has three mandatory files: .shp, which contains the actual feature geometry; .shx, which provides a positional index of the feature geometry; and .dbf, which holds attributes for each shape.

reference

4.2.2. SQL Database

The second method used is to load the data into a spatial SQL database and provide the model with database schemas that can be used to generate queries. The datasets were uploaded to a dockerized PostGIS database using QGIS's DB Manager plugin.

Some of the datasets come in the GML data format, which can include multiple layers with potentially different geometries. For this reason, they cannot be loaded directly into a PostGIS database such that they are stored in the same database table. Furthermore, several of the layers in the multi-layer GML files are irrelevant for most analysis situations. For instance, the flood zone data were downloaded as a multi-layer GML file and includes a total of eight layers: polygon and multi-line border for the analysis area, polygon layers for rivers, ocean surfaces, and lakes, polygon and multi-line border for the flood zones, and a layer containing cross-sectional profile lines for the rivers. The quick clay dataset was similar. For brevity, a decision was made not to load all these layers into the PostGIS database. Only the polygon for the flood zones and two polygon layers from the quick clay dataset were loaded into the database.

4.2.3. OGC API Features

The third method for data access is to use the OGC API Features standard.

5. Architecture

5.1. High-Level Application Architecture

A microservice architecture was employed in order to simplify development and separate concerns between the different microservices. The services are deployed as Docker Containers, and they are orchestrated using Docker Compose. Figure 5.1 shows how the application is divided into five distinct services, and the direction of information flow between these.

quickly explain docker and why it is useful here

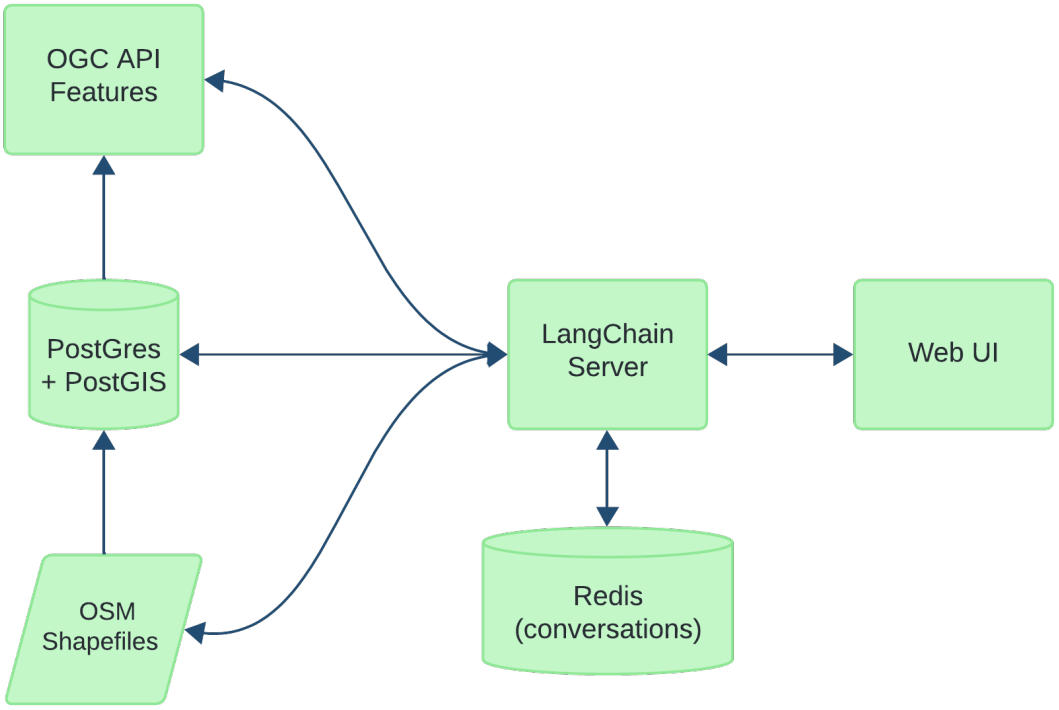


Figure 5.1.: Architecture overview

5.1.1. LangChain Server

The *LangChain Server* service is the heart of the application, and is where the Large Language Model (LLM)-related logic is situated. It is responsible for taking requests from

the *Web UI* and returning suitable responses in what becomes a client-server architecture between the two services. Table 5.1 show the endpoints exposed by the server and how they can be used by a client.

Table 5.1.: Summary of Server Endpoints

Endpoint	Method	Description
/session	GET	Takes a `session_id` as a query parameter, allowing the client to continue on a pre-existing session.
/session	POST	Creates a new session with an empty conversation.
/streaming-chat	GET	Endpoint for chatting the LLM. Takes a `message` as a query parameter and returns an event stream, allowing for token streaming from server to client.
/update-map-state	POST	Send the state of the client map to the server. Keeps the server updated on what layers are present in the map, their color, etc.
/geojson	GET	Takes a `geojson_path` as a query parameter. Allows the client to retrieve a given GeoJSON file that is stored in the working directory on the server.
/history	GET	Used to retrieve the chat history of the current session.
/upload	POST	Allows the client to upload one or more files to the working directory on the server.

5.1.2. Redis for Conversations

Redis (Sanfilippo, 2009) is a fast in-memory database that is often applied as a caching database that sits on top of some persistent database. It can also be used for vector-based storage and as a simple NoSQL database. The latter option is the way it is used in GeoGPT's architecture, and its only purpose is to store conversations. Whenever a user starts a conversation with GeoGPT an object with a unique session ID is stored to the Redis database. This object holds an array that represents the conversation. This array is written to every time either the human or GeoGPT produces a message.

Storing messages — either in memory as a simple array or in a database like Redis — is crucial to enable multi-message conversations. In order for a LLM to act as a conversational agent, some sort of chat history needs to be prepended to the prompt. In the case of GeoGPT the entire chat history is prepended. This has the advantage of providing the LLM with the complete context of the chat history, but the disadvantage of potentially bloating the context window from which it is supposed to generate tokens.

Therefore, as the chat becomes longer each new token will be both more expensive and take more time to get generated. A long chat history could also make the resulting prompt exceed the token limit of the LLM, or it could confuse model by providing it with messages no longer relevant to the conversation. These issue was not considered in great detail for this project, and are left for future work.

5.1.3. PostGres + PostGIS

5.1.4. OGC API Features

On top of the PostGIS database containing OpenStreetMap (OSM) data is a RESTful geospatial feature server called *pg_featureserv* (CrunchyData, 2024).

5.1.5. Web UI

The user interface is made with SolidJS. By design, it is very minimal. One of the goals of the project is to simplify the way we do GIS analysis. One of the key design goals was therefore to make the interface as familiar to the user as possible and lowering the chance of the user doing something wrong. The chat interface was designed to imitate the interface of OpenAI's

5.2. Agent Architecture

Three different agent were implemented for GeoGPT.

Subsection 5.2.1 will explain how the base agent was implemented using LangGraph, while subsection 5.2.3 will present a detailed description of how GeoGPT builds suitable prompts that provide the LLM with the context required to solve the problem. Finally, subsection 5.2.2 will present the different tools that were used for the different agents.

5.2.1. LangGraph Agent Implementation

5.2.2. Tools

5.2.3. Prompt Templating

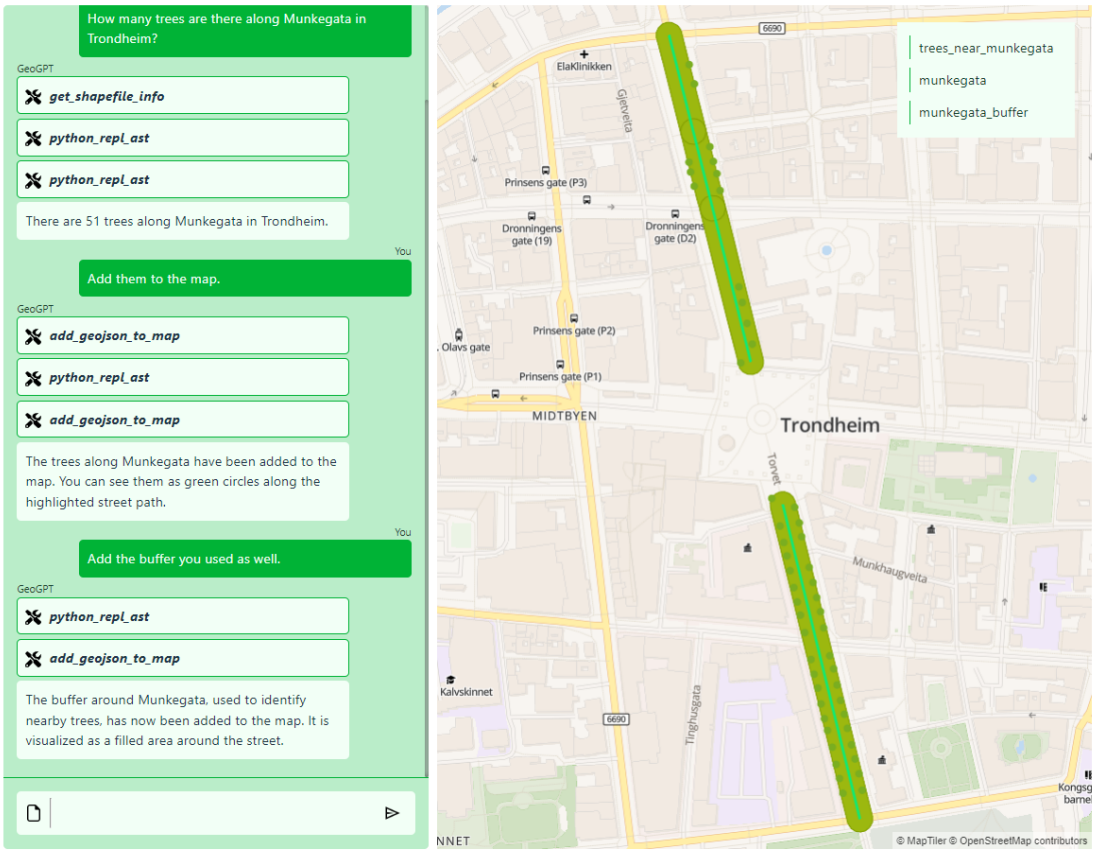


Figure 5.2.: Web UI

Table 5.2.: Overview of each agent's tools

Agent Type	**Tools**

OGC API Features	`get_collections_info`
	`list_collections`
	`query_collection`
	`python_repl_ast`
	`add_geojson_to_map`

Python	`python_repl_ast` `add_geojson_to_map`

SQL	`sql_db_query`
	`sql_db_list_tables`
	`sql_db_schema`
	`add_geojson_to_map`

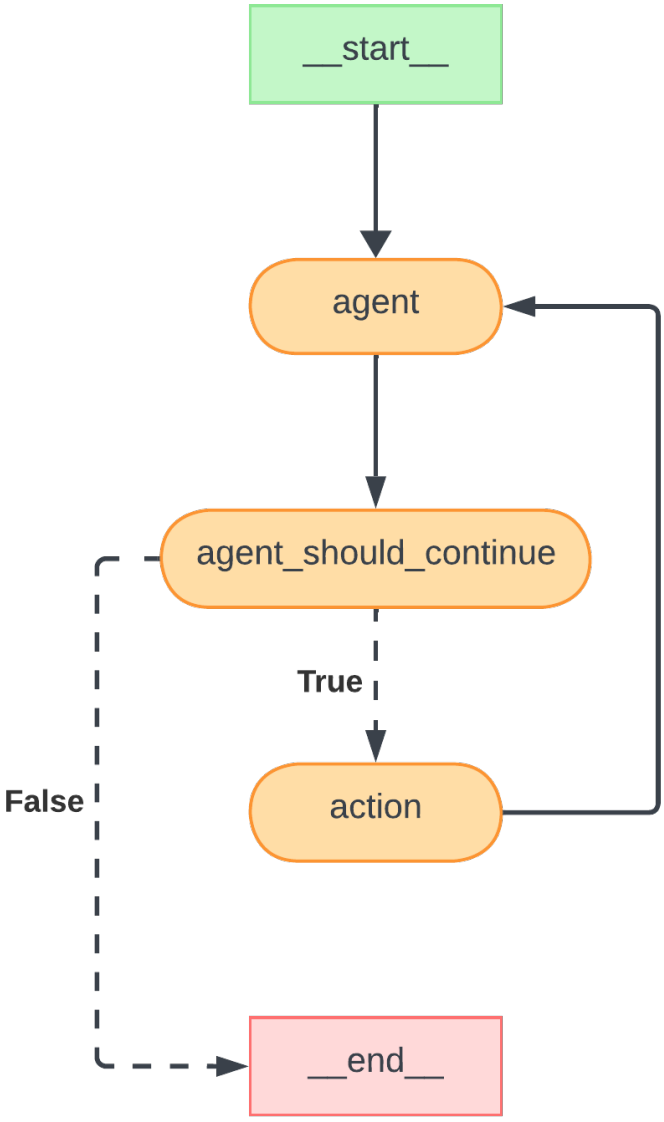


Figure 5.3.: Generic tool agent graph

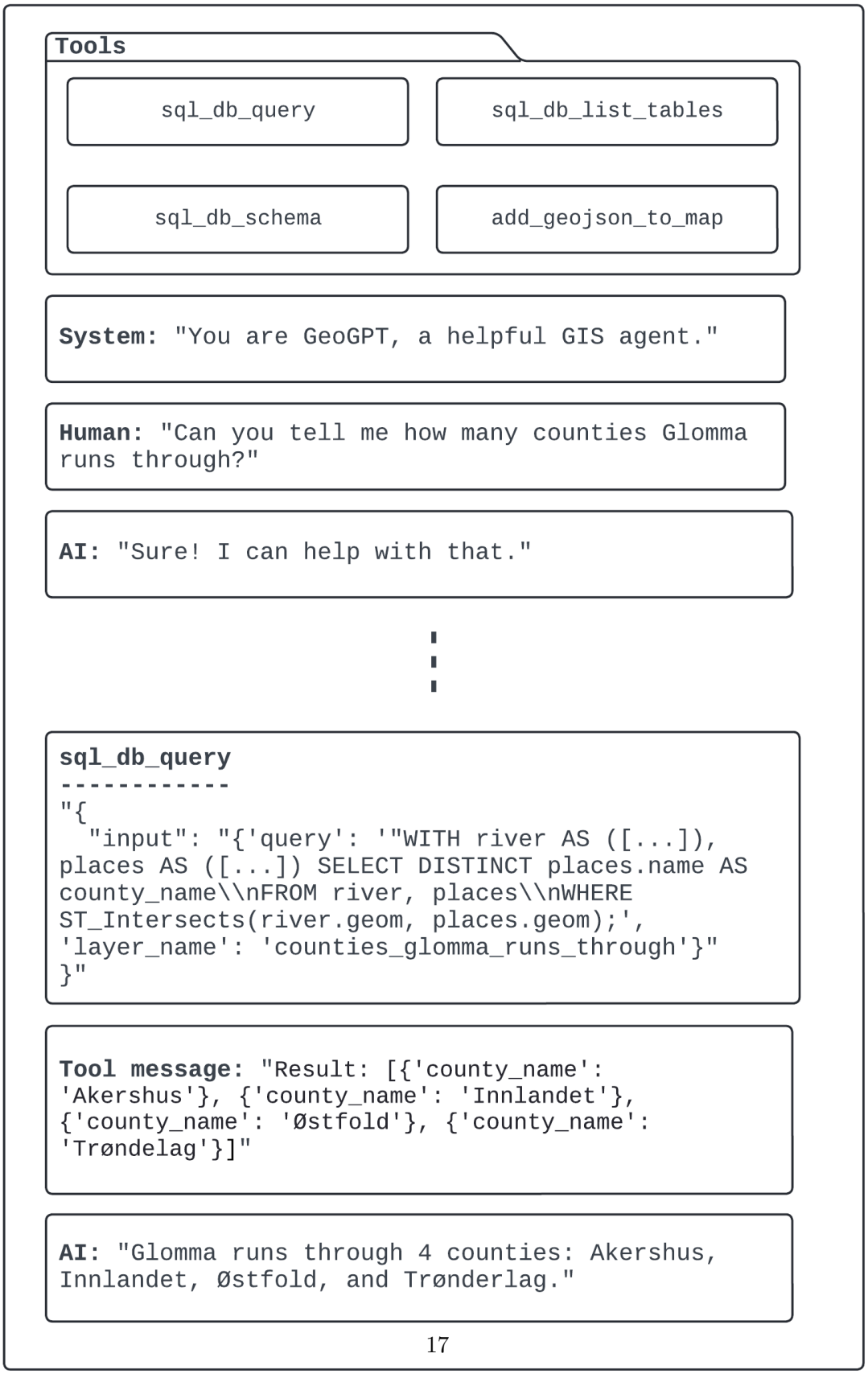


Figure 5.4.: Example of a chat trace

6. Experiments and Results

6.1. Experimental Setup

The experiments conducted to evaluate the performance of GeoGPT on geospatial tasks are divided into three main approaches, as elaborated upon in

6.1.1. Quantitative Approach: Benchmarking

The first approach seeks to evaluate its ability to successfully answer questions that have a concrete answer. To do this, a Q&A dataset was constructed. This dataset consists of set of 12 GIS-related questions with corresponding correct answers. For each record in the dataset, a description of how a human would find it natural to approach the problem. This description is provided as a step-by-step path towards the solution, and is only included to guide any reader as to how the system would be expected to solve the system. The full Q&A dataset can be found in Appendix B. This set of experiments will allow for quantitative assessment of GeoGPT's GIS-abilities, and is a feasible way of benchmarking the system.

Another aspect that the benchmarking approach will try to evaluate is the consistency of the system, its ability to repeatedly provide an acceptable answer to the same user question. Each of the 12 questions are therefore asked three times per agent type. With the implementation of three different agent types, the total number of test runs becomes the following:

$$12 \text{ questions} \cdot 3 \text{ agent types} \cdot 3 \text{ repetitions} = 108 \text{ tests}$$

Each test run's answer will be manually evaluated, and the outcome will be annotated as one of *success*, *partial success*, and *failure*. Table 6.1 shows the guidelines used when assigning test results.

The application is hooked up to LangChain AI's tracing system, *LangSmith*. Apart from being a useful tool for debugging purposes, it provides a simple way of obtaining detailed data for token and time usage for a particular run. These are metrics that will be recorded and used in the evaluation of GeoGPT. To summarize, the following metrics are recorded for a given test run:

- The outcome of the test (*success*, *partial success*, or *failure*)
- The total duration in seconds
- The total number of tokens used

Table 6.1.: Description of Success

Outcome	Guideline
Success	The question was answered correctly and little to no follow-up from the user was required to produce the desired outcome. No false assumptions were made by the system when answering the question.
Partial Success	Portions of the question were answered correctly or semi-correctly, and/or some follow-up from the user was required to guide the system toward the solution.
Failure	The question was answered incorrectly answered and/or false assumptions were made by the system while attempting to answer the question.

- The total cost for the run in American dollar

Subsection 6.2.1 will present the results of the benchmarking tests.

6.1.2. Evaluating Importance of Initial Prompt

The second set of experiments are constructed to evaluate the importance of the initial question/prompt from the human user. As stated in Background and Motivation, part of the motivation for developing an LLM-driven GIS like GeoGPT is to make GIS more accessible to non-experts. At the same time, it may be valuable to assess the extent to which a carefully constructed prompt by a GIS expert can enhance the system's output.

6.2. Experimental Results

It is worth noting that two slightly different models were used during testing. The explanation of this is the release of the `gpt-4-turbo-2024-04-09` in mid-April. According to OpenAI, "this new model is better at math, logical reasoning, and coding" compared to `gpt-4-0125-preview`[1], which is the model that was used at the start of the experimentation phase of the master's thesis. At the new model's release, a decision was made to use this for the remaining experiments. The experiments that had already been conducted were not re-run due to time constraints and a belief that these slight model upgrades would not significantly change the outcome of the experiments.

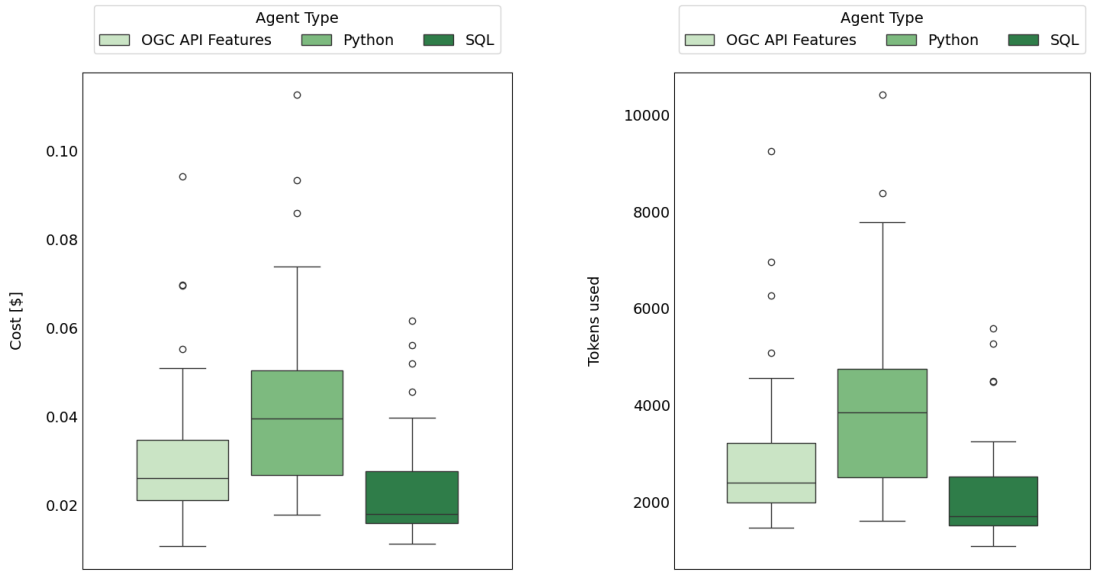
[1]OpenAI has a GitHub repository containing the code they use to evaluate their Large Language Models (LLMs) and benchmark results for OpenAI models and reference models from other companies: https://github.com/openai/simple-evals.

6.2.1. Quantitative Results

Figure 6.1.: Outcome Distribution

Table 6.2.: Encoding for Test Outcome

Outcome	**Encoded Value**
Success	2
Partial Success	1
Failure	0



(a) Average cost per call for each agent type

(b) Average token usage per agent type

Figure 6.2.: Cost and token usage

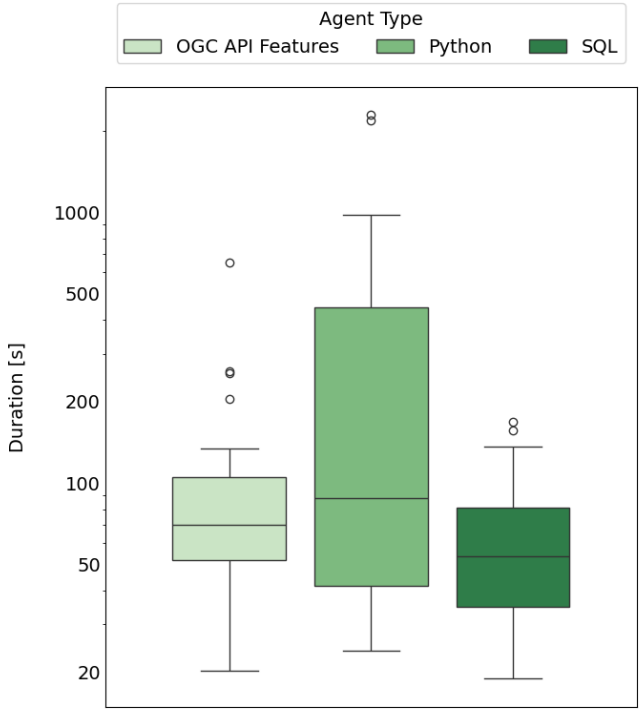


Figure 6.3.: Duration per Agent Type

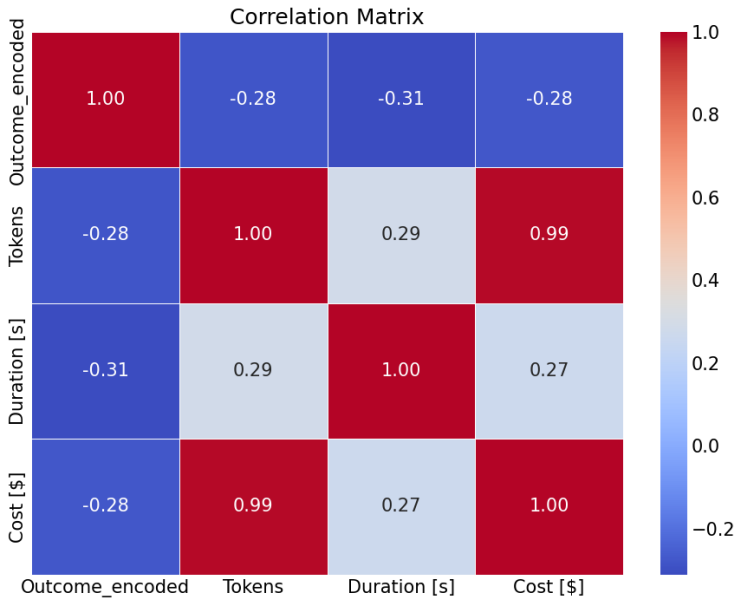


Figure 6.4.: Duration per Agent Type

Table 6.3.: Standard Deviation by Agent Type

Agent Type	**Outcome Std. Deviation**
OGC API Features	0.552
Python	0.337
SQL	0.337
Mean	0.408

7. Evaluation and Discussion

7.1. Evaluation

7.2. Discussion

8. Conclusion and Future Work

8.1. Contributions

8.2. Future Work

8.2.1. Automated Data Access

The experiments in chapter 6 were based upon a pre-existing database. A fully autonomous GIS agent should, however, be able to search the web for suitable datasets, based on the user's query. In a Norwegian context, one could imagine asking for a noise analysis for a particular building. The agent would then search Geonorge for datasets related to noise (firing ranges, roads, etc.), downloading these, and then performing analysis. Simple experiments were conducted in this thesis to see if this was possible, but results were somewhat poor. Methods like semantic search based upon the documentations of datasets should be explored in future research.

Bibliography

CrunchyData. (2024). CrunchyData/pg_featureserv. Retrieved April 19, 2024, from https://github.com/CrunchyData/pg_featureserv

Eleti, A., Harris, J., & Kilpatrick, L. (2023). Function calling and other API updates. Retrieved March 10, 2024, from https://openai.com/blog/function-calling-and-other-api-updates

Holm, O. (2023). *LLMs - The Death of GIS Analysis?* (Specialization Project). NTNU. Trondheim. https://github.com/oskarhlm/prosjektoppgave/blob/main/tex/auxiliary/main.pdf

Hong, S., Zhuge, M., Chen, J., Zheng, X., Cheng, Y., Zhang, C., Wang, J., Wang, Z., Yau, S. K. S., Lin, Z., Zhou, L., Ran, C., Xiao, L., Wu, C., & Schmidhuber, J. (2023). MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework. https://doi.org/10.48550/arXiv.2308.00352

LangChain AI. (2022). Langchain-ai/langchain. Retrieved October 5, 2023, from https://github.com/langchain-ai/langchain

LangChain AI. (2024). Langchain-ai/langgraph. Retrieved March 21, 2024, from https://github.com/langchain-ai/langgraph

OpenAI. (2022). Introducing ChatGPT. Retrieved October 26, 2023, from https://openai.com/blog/chatgpt

Radford, A., & Narasimhan, K. (2018). Improving Language Understanding by Generative Pre-Training. Retrieved October 9, 2023, from https://www.semanticscholar.org/paper/Improving-Language-Understanding-by-Generative-Radford-Narasimhan/cd18800a0fe0b668a1cc19f2ec95b5003d0a5035

Sanfilippo, S. (2009). Redis - The Real-time Data Platform. Retrieved April 19, 2024, from https://redis.io/

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention Is All You Need. Retrieved October 10, 2023, from https://arxiv.org/abs/1706.03762v7

Appendices

A. Task Description from Norkart

Oppgåve med omfang som kan tilpassast både prosjekt og masteroppgåve

LLMs - GIS-analysens død

(kan justerast seinare)

BAKGRUNN

Nyere modeller for kunstig intelligens har demonstrert spesielt gode evner til å kunne lære av store mengder ustrukturert og semi-strukturert informasjon. ChatGPT fra OpenAi tok verden med storm – og chat-baserte systemer florerer. Kan chat-baserte modeller skapes for å hente ut GIS-data effektivt? Norkart har en stor dataplattform hvor brukere utvikler mot API'er som i stor grad har GIS/Geografiske data i bunn. GeoNorge er en stor datakatalog hvor brukere slår opp, eller søker kategorisert for å finne data. QGIS, Python, PostGIS, FME og andre verktøy brukes ofte til å gjennomføre GIS-analyser – hvor en GIS-analytiker/data-scientist gjennomfører dette.

«*Finn alle bygninger innenfor 100-meters-belte som er over 100 kvm og har brygger*»

Er dette mulig å få til med dagens tilgjengelige chat-modeller?

OPPGAVEBESKRIVELSE

Oppgaven har som hovedmål å undersøke hvordan nyere språkmodeller kan benyttes for å gjennomføre klassiske GIS-analyser ved å bruke standard GIS-teknologi som PostGIS/SQL og datakataloger (OGC API Records fks). Hva finnes av tilgjengelig chat-løsninger? Hvordan spesialtilpasse til GIS-anvendelser? Hvor presise kan en GIS-Chat bli?

Relevante delmål for oppgaven:

1. Kartlegge state-of-the-art
2. Utvikle proof-of-concepts
3. Analysere begrensninger og kvalitet

Oppgaven vil med fordel deles i prosjektoppgave og masteroppgave

- Prosjektoppgave
  - State-of-the-art: Ai-modeller og multi-modal maskinlæring
  - Innhente og utvikle datagrunnlag og API-tilgjengelighet
- Masteroppgave
  - Utvikle proof-of-concepts med tilgjengelige åpne modeller/teknologi
  - Gjennomføre eksperimenter for analyse av kvalitet

NTNU
Faculty of Engineering Science and Technology
Department of Civil and Environmental Engineering

Detaljert oppgavebeskrivelse utvikles i samarbeid med studenten.

ADMINISTRATIVT/VEILEDNING

Ekstern veileder: (en eller flere)
Mathilde Ørstavik, Norkart
Rune Aasgaard, Norkart
Alexander Nossum, Norkart

Aktuelle vegleiarar og ansvarleg professor ve NTNU (den som har fagansvar nærast oppgåva):

Terje Midtbø (GIS, kartografi, visualisering)
Hongchao Fan (3D modellering, fotogrammetri, laser)

B. Q&As for Benchmark

Question	Difficulty	Answer	Reasoning
Find all buildings that are in the danger zone of a being flooded, in accordance to current regulations for new buildings.	2	Dataset containing building points: ...\data\bygninger_-100årsflom.shp	Steps: 1. Realize that new buildings should be secure against 200-year floods. 2. Gather and load flood risk data and building point data. 3. Ensure that both datasets are in the same coordinate reference system (CRS). 4. Filter out the buildings that are located within the 200-year flood zone.

Find all buildings that are in the danger zone of a being flooded, in accordance to current regulations for new buildings.	2	Dataset containing building points: ...\data\bygninger_-100årsflom.shp	Steps: 1. Realize that new buildings should be secure against 200-year floods. 2. Gather and load flood risk data and building point data. 3. Ensure that both datasets are in the same coordinate reference system (CRS). 4. Filter out the buildings that are located within the 200-year flood zone.