RideWise: Enhancing Biking Journeys with LLM-Assisted Navigation

Nick Gardner

Advisor: Dr. Zachary Butler

Computer Science @ RIT
B. Thomas Golisano College of Computing & Information Sciences

Introduction

Modern navigation apps are extremely effective for cars, but are missing the adaptability necessary for bicycling. In this work, I have attempted to address these shortcomings by incorporating a *Large Language Model* (LLM) into the standard navigation app structure. This element allows users to request custom modifications in a free-form and easy-to-use format.

Objectives

- Develop a solution for more effective bicycle navigation.
- Learn about LLM prompting techniques and system design strategies.
- Compare effectiveness of open-source LLMs vs. commercial models (ChatGPT), as well as single models vs. ensemble.

Client

The client is the interface that the user interacts with. On opening the web page, the user is greeted with a familiar navigational setup: input boxes for the start and destination, as well as a dynamic map. These boxes are connected to the Google Maps API, allowing for region-biased autocompletions. The dynamic map is rendered by the same API service, ensuring consistent results. Unique to this application, there is an additional input box for submitting text-based route requests. Accompanying this text box are two buttons, one for initiating speech-to-text capabilities, and one for clearing current route modifications. The page is rendered using HTML, with JavaScript to handle interactivity and Bootstrap CSS for styling. When the route is received from the Flask server, it is rendered as GeoJSON on the data layer of the dynamic map.

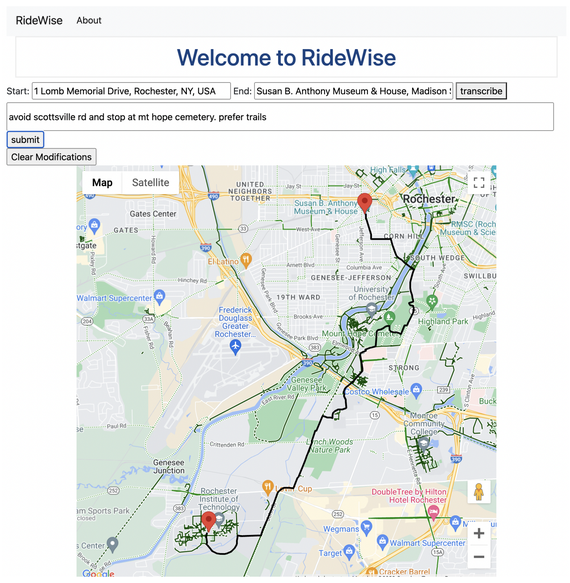

Figure: Application interface with example route and modifications.

System Architecture

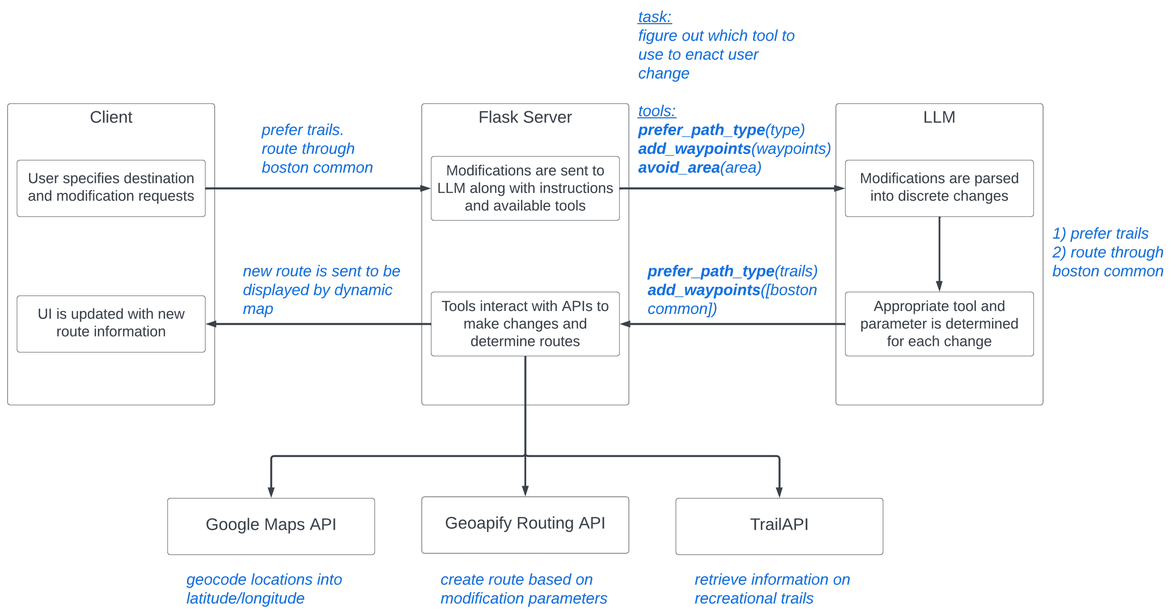

Figure: System architecture diagram. Marked in black are system components and connections. Marked in blue is an example input and corresponding system actions.

LLM

The Llama 2 family of models were selected for study, as these models are state-of-the-art in open-source. Specifically, a quantized C++ distribution of the 13B-chat model is used, as this is the largest version of the fine-tuned chat models that fit in system memory.

Two tasks are fulfilled by the LLM:

- **Splitting Input**
  1. *I want to ride on trails and route through central park and the empire state building. don't go on stevenson ave.*
  2. *I want to ride on trails | route through central park and the empire state building | don't go on stevenson ave*
- **Choosing Change**
  1. *route through central park and the empire state building*
  2. *add_waypoints | central park | empire state building*

These tasks are relatively easy for commercial models, but require careful prompt engineering for open-source models to be effective. This technique is known as *few-shot prompting*, where the model is not fine-tuned on domain data but instead is shown a small number of examples of successful operation as part of its prompt. The effectiveness of this technique demonstrates the uniqueness of pre-trained LLMs – rather than having to construct a large dataset to train (and likely overtrain) a model, we simply prime the model and rely on its language parsing capabilities.

Server

Even with perfect LLM execution, this application is not useful if the requested changes are not successfully applied to the route. With this in mind, the server uses a series of API calls to safely and correctly generate the route:

1. Waypoints are geocoded (location string converted to latitude/longitude) and checked for distance. If a waypoint fails either test, it is discarded.
2. Path type is checked to see if it contains various key words that indicate different operational modes – {bicycle, mountain_bike, city_bike}.
3. If new locations to avoid are present, a new route is generated with waypoints, path type, and previous avoids.
4. Next, the legs of the route are explored to determine if the avoid string represents the name of any of the paths on the route ("scottsville road" for example). If so, then the start and end coordinates of that step are used as avoid waypoints.
5. If not, then the avoid location is geocoded. If it cannot be geocoded or it is too far away, it is discarded.
6. The request string is constructed with the processed input parameters and sent to Geoapify which returns a GeoJSON route.

Results

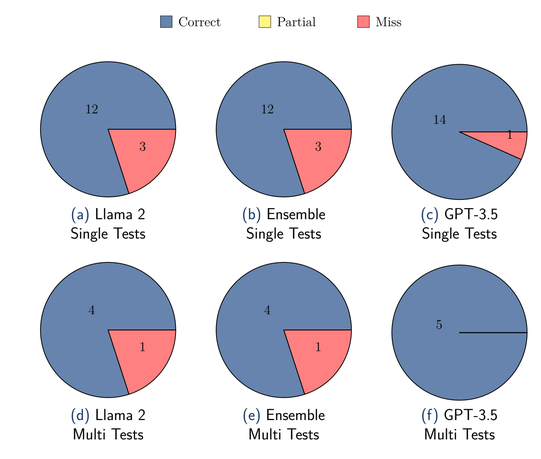

Figure: Route modification evaluation on a hand-created dataset of 15 "single" route modifications (where only one style of change is requested) and 5 "multi" route modifications (where all three styles of change are requested).

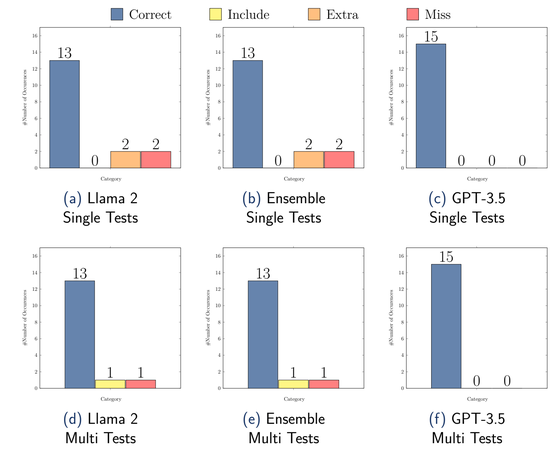

Figure: Correctness evaluation for LLM processing. The category "include" indicates that the model output included the necessary changes, but also included additional incorrect items. "extra", for single tests only, indicates that items were included for categories not part of the test.

Conclusion

A functional application was constructed that allows for more effective bicycle navigation. Additionally, study and employment of cutting-edge LLM techniques brought open-source model performance to a similar level as existing commercial models on this task. Future work could consider expanding change request capabilities, improving the UI, and revisiting the LLM implementation with forthcoming advancements in open-source LLMs.