Multimodal Data Analysis with LLms and Python - Tutorial

Course developed by Immanuel Trummer, PhD

Luka Bostick
University of North Texas for HackUNT 24
LukaBostick@my.unt.edu

September 15, 2024

L B T M

Research and Development

Contents

0.1 Introduction 1
0.2 Setup 1
0.3 Analyzing Text 2

Acknowledgments

Thanks to Dr. Timothy M. Pinkston, Dr. Murali Annavaram, and Dr. Sasan Razmkhah for providing me with a voice and a potential platform to share my input and thoughts on superconducting single flux quantum technology.

Special thanks to the review committee:

0.1 Introduction

Large language models have recently revolutionized various areas of computer science including text processing code generation and so on. The newest generation of those language models such as OpenAI's o1-preview[1], and GPT4 Omn [2] is multimodel; meaning that they can process not only text input but also various other types of data, for example Images or sounds. In this tutorial Dr. Trummer shows how you can use Large Language models and Python[3] in order to analyze various different types of data including tabular text processing, analyzing images, and analyzing sound data all from the same python API

0.2 Setup

Install Openai

We will be using openai's language models in this tutorial, there are many other providers of Large language models. The interfaces ten to be quite similar, but in this tutorial we are going to focus on the openai model. In order to use this model from python we install the corresponding python library as show below

```
@LBTM-Luka → /workspaces/LukaBostick (main) $ pip install openai
Collecting openai
  Downloading openai-1.45.0-py3-none-any.whl.metadata (22 kB)
Requirement already satisfied: anyio<5,>=3.5.0 in /home/codespace/.local/lib/python3.12/site-packages (from openai) (4.4.0)
Collecting distro<2,>=1.7.0 (from openai)
  Downloading distro-1.9.0-py3-none-any.whl.metadata (6.8 kB)
Requirement already satisfied: httpx<1,>=0.23.0 in /home/codespace/.local/lib/python3.12/site-packages (from openai) (0.27.0)
Collecting jiter<1,>=0.4.0 (from openai)
  Downloading jiter-0.5.0-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (3.6 kB)
Collecting pydantic<3,>=1.9.0 (from openai)
  Downloading pydantic-2.9.1-py3-none-any.whl.metadata (146 kB)
Requirement already satisfied: sniffio in /home/codespace/.local/lib/python3.12/site-packages (from openai) (1.3.1)
Collecting tqdm>4 (from openai)
  Downloading tqdm-4.66.5-py3-none-any.whl.metadata (57 kB)
Collecting typing-extensions<5,>=4.11 (from openai)
  Downloading typing_extensions-4.12.2-py3-none-any.whl.metadata (3.0 kB)
Requirement already satisfied: idna>=2.8 in /home/codespace/.local/lib/python3.12/site-packages (from anyio<5,>=3.5.0->openai) (3.7)
Requirement already satisfied: certifi in /home/codespace/.local/lib/python3.12/site-packages (from httpx<1,>=0.23.0->openai) (2024.7.4)
Requirement already satisfied: httpcore==1.* in /home/codespace/.local/lib/python3.12/site-packages (from httpx<1,>=0.23.0->openai) (1.0.5)
Requirement already satisfied: h11<0.15,>=0.13 in /home/codespace/.local/lib/python3.12/site-packages (from httpcore==1.*->httpx<1,>=0.23.0->openai) (0.14.0)
Collecting annotated-types>=0.6.0 (from pydantic<3,>=1.9.0->openai)
  Downloading annotated_types-0.7.0-py3-none-any.whl.metadata (15 kB)
```

We now see that the library has already been installed to confirm insulation and to check the version of the library do the following below

[1] https://openai.com/index/introducing-openai-o1-preview/
[2] https://openai.com/index/hello-gpt-4o/
[3] https://www.python.org/

```
@LBTM-Luka → /workspaces/LukaBostick (main) $ pip show openai
Name: openai
Version: 1.45.0
Summary: The official Python library for the openai API
Home-page: https://github.com/openai/openai-python
Author:
Author-email: OpenAI <support@openai.com>
License:
Location: /usr/local/python/3.12.1/lib/python3.12/site-packages
Requires: anyio, distro, httpx, jiter, pydantic, sniffio, tqdm, typing-extensions
Required-by:
@LBTM-Luka → /workspaces/LukaBostick (main) $
```

depending on when you are doing this tutorial the version may differ. For function interoperability it might be best to install precisely the same version via the following below:

```
@LBTM-Luka → /workspaces/LukaBostick (main) $ pip install openai==1.29
Collecting openai==1.29
  Downloading openai-1.29.0-py3-none-any.whl.metadata (21 kB)
Requirement already satisfied: anyio<5,>=3.5.0 in /home/codespace/.local/lib/python3.12/site-packages (from openai==1.29) (4.4.0)
Requirement already satisfied: distro<2,>=1.7.0 in /usr/local/python/3.12.1/lib/python3.12/site-packages (from openai==1.29) (1.9.0)
Requirement already satisfied: httpx<1,>=0.23.0 in /home/codespace/.local/lib/python3.12/site-packages (from openai==1.29) (0.27.0)
Requirement already satisfied: pydantic<3,>=1.9.0 in /usr/local/python/3.12.1/lib/python3.12/site-packages (from openai==1.29) (2.9.1)
Requirement already satisfied: sniffio in /home/codespace/.local/lib/python3.12/site-packages (from openai==1.29) (1.3.1)
Requirement already satisfied: tqdm>4 in /usr/local/python/3.12.1/lib/python3.12/site-packages (from openai==1.29) (4.66.5)
Requirement already satisfied: typing-extensions<5,>=4.7 in /usr/local/python/3.12.1/lib/python3.12/site-packages (from openai==1.29) (4.12.2)
Requirement already satisfied: idna>=2.8 in /home/codespace/.local/lib/python3.12/site-packages (from anyio<5,>=3.5.0->openai==1.29) (3.7)
Requirement already satisfied: certifi in /home/codespace/.local/lib/python3.12/site-packages (from httpx<1,>=0.23.0->openai==1.29) (2024.7.4)
Requirement already satisfied: httpcore==1.* in /home/codespace/.local/lib/python3.12/site-packages (from httpx<1,>=0.23.0->openai==1.29) (1.0.5)
Requirement already satisfied: h11<0.15,>=0.13 in /home/codespace/.local/lib/python3.12/site-packages (from httpcore==1.*->httpx<1,>=0.23.0->openai==1.29) (0.14.0)
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/python/3.12.1/lib/python3.12/site-packages (from pydantic<3,>=1.9.0->openai==1.29) (0.7.0)
Requirement already satisfied: pydantic-core==2.23.3 in /usr/local/python/3.12.1/lib/python3.12/site-packages (from pydantic<3,>=1.9.0->openai==1.29) (2.23.3)
Downloading openai-1.29.0-py3-none-any.whl (320 kB)
Installing collected packages: openai
  Attempting uninstall: openai
    Found existing installation: openai 1.45.0
    Uninstalling openai-1.45.0:
      Successfully uninstalled openai-1.45.0
Successfully installed openai-1.29.0
```

After installing the library you still need to make sure that you openai account[4]
After logging in (or creating an account), you can preform the steps that i'm going to show in the following

0.3 Analyzing Text

In this section we will use LLM to analyze text in order to classify test documents based on whether or not the underlying sentiment is positive or negative. For a more concrete example imagine that you have a couple of reviews that are provided as pure text then count the revise that are positive and the reviews that are negative in order to see an accurate evaluation of performance. This type problem is also called sentiment classification. meaning that our input is a text document → tokenized → then re-classified as a sentiment score for the output. In the rest of this section we will learn to implement sentiment classification using Python using the GPT4 Omni model.
Let us review the setup one last time before moving on.

1. Install Python
2. Install the openai library
3. Create Api Key
4. Specified openai access key in an environment variable

The first thing we need to do in order to use Open models from python is demonstratied below:

[4] https://platform.openai.com/docs/overview

```
'''
Created on Sep 15, 2024

@Author immanueltrummer & Luka Bostick

'''

import openai
```

The seconc thing that you want to do when using the openai model is to create a client object. A client object is how you interact with the openai library.

```
'''
Created on Sep 15, 2024

@Author immanueltrummer & Luka Bostick

'''

import openai

client = openai.OpenAI();
```

Above we have already specified the openai access key in an environment variable. If we have not done that we could pass in the access key as a formal parameter for the client object, this is not recommend.

```
'''
Created on Sep 15, 2024

@Author immanueltrummer & Luka Bostick

'''

import openai

client = openai.OpenAI();

if__name__ == '__main__';
```

The code above ensurtes that the python file will execute if directly accesses by our IDE.
Below we will assume that users provide the review to classify as an input parameter, in order to process these input parameters we import the following input as well as a parser instance. Next add the arguments we want to parse, it should look the following:

```
'''
Created on Sep 15, 2024

@Author immanueltrummer & Luka Bostick

'''
import argparse
import openai

client = openai.OpenAI();

if__name__ == '__main__';

  parser = argparse.ArgumentParser()
```

Now we add arguments that we want to parse from the command line so, it is going to be text that we want to classify based on whether it's positive or negative sentiment so the type of this input argument as the is a string that will be shown to users if they don't provide this

input parameter and here that's simply a text to classify all, so here in going to pass the input arguments this should look like the following

```
'''
Created on Sep 15, 2024

@Author immanueltrummer & Luka Bostick

'''
import argparse
import openai

client = openai.OpenAI();

if__name__ == '__main__';

  parser = argparse.ArgumentParser()
  parser.add_argument('text', type=str, help ='A text to classify')
  args = parser.parse_args()
```

So now this args object now contains all the value for those input parameters, now whenever you use a language model you have to provide it with a prompt as input. The prompt essentially describes the task that we want the language model to solve so more case the task is to classify the input text in terms of whether or not the underlying sentiment is positive.
Now let me introduce a new function that creates the prompts that we want to send to the language model. So here the below the prompt will essentially consist of three parts

1. Provide the input text to the language model
2. Provide the language model with some instructions (On what it should do with the input)
3. Provide formatting instructions with regards to the answer

Bellow is the following function *create_prompt*, our formal parameter will be text. The argument that we are passing is the text we want to classify.
What it return is a prompt "for text classification
Next we want to provide some instructions. There are a couple of different ways to answer this question. I could be an entire sentence as answer maybe the language model writes back, under laying sentiment is positive or maybe the language decides to abbreviate positive by

$$P$$

and negative by

$$n$$

so in order to accomplish this we want to provide a little bit of formatting for the language model. Then return with the signature parameters; first we output or formal parameter '*text*' then the formatting '*instructions*' then we encourage the language model to to come up with an answer.

```
'''
Created on Sep 15, 2024

@Author immanueltrummer & Luka Bostick

'''
import argparse
import openai

client = openai.OpenAI();
```

```
11
12 def create_prompt(text):
13   """ Create input prompt for the language model.
14
15   Args:
16        text: text to classify.
17
18   Returns:
19        prompt for text classification.
20   """
21         instructions = 'Is the underlying sentiment positive or negative?'
22         formatting = '"Positive or Negative"'
23         return f'Text:{text}\n{instructions}\nAnswer({formatting}):'
24
25 __name__ == '__main__';
26
27   parser = argparse.ArgumentParser()
28   parser.add_argument('text', type=str, help ='A text to classify')
29   args = parser.parse_args()
```

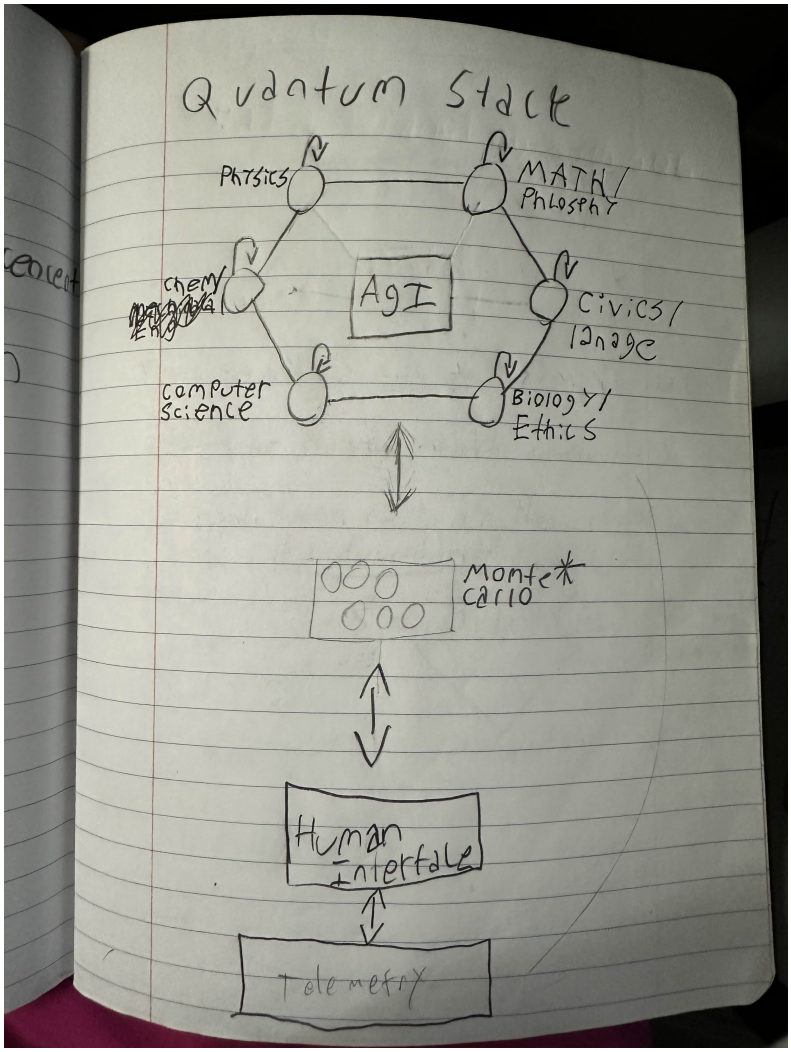
All right so this is a concise prompt which describes to the language model what we wanted to do in natural language of course, and it contains all relevant context namely the formatting instructions for the output as well as the input text that we want to classify.

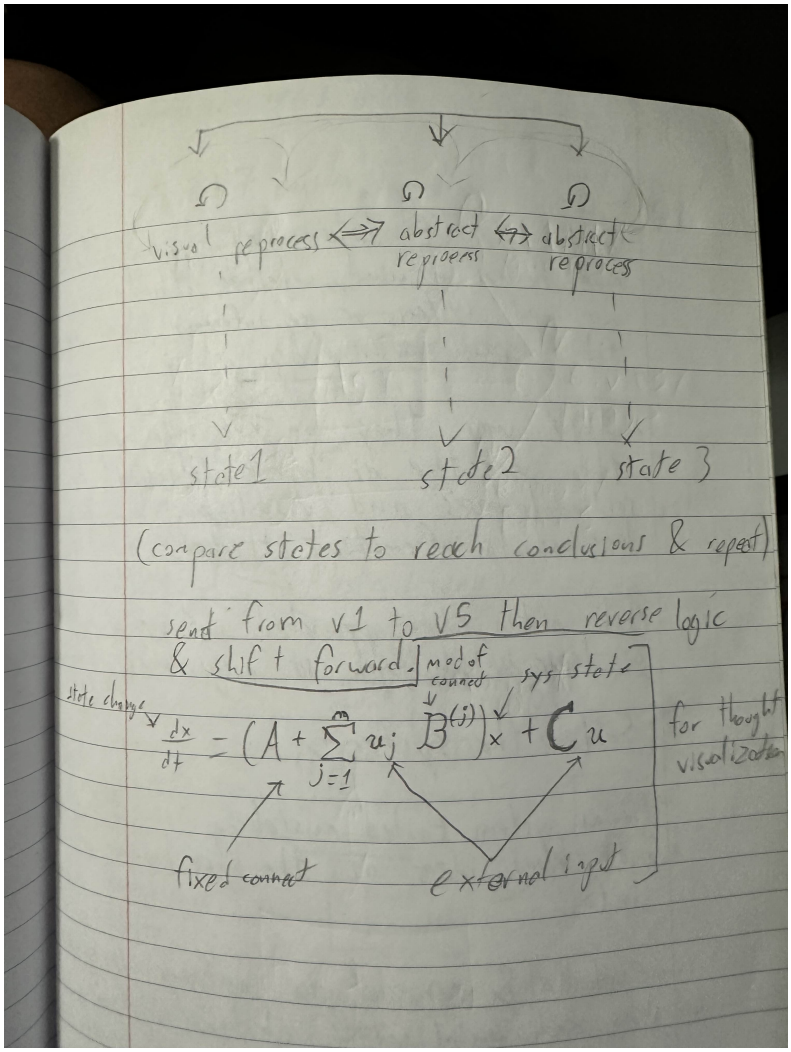
Next we will add a way of creating a prompt so I could for instance instantiate a prompt variable in order to create a corresponding prompt, then use that the parameter '*text*' that the users have provided, the next thing we need to do is call that language model with that prompt, so we can generate prompts the next step is to send those prompts to a language model in order to get a corresponding reply so in the following I'm going to introduce the function **call_llm**

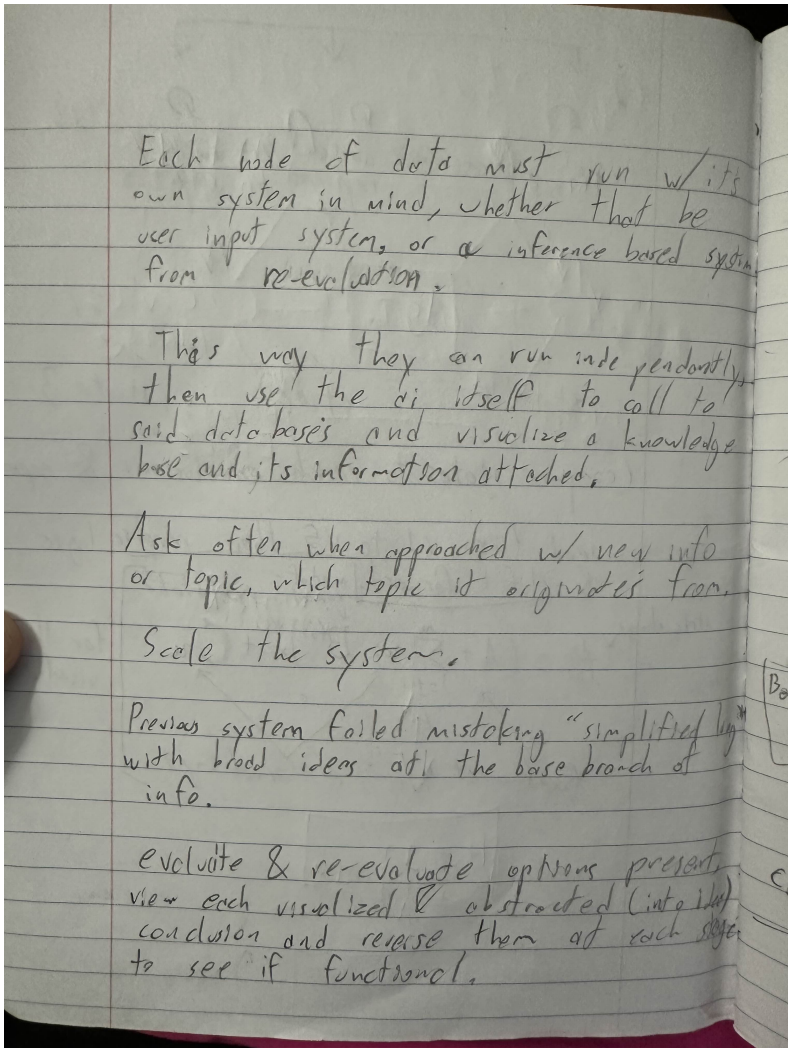
This function takes the formal parameter and appends documentation to the param. The goal of this function is to call the large language model with the input prompt in order to instruct it to be sent to the large language model. And the function returns the answer that it obtains from the language model.

After we call the LLM we next use the client object that we have introduced before. We will use the Openai GPT4 model[5] (the most available model released by the writing of this tutorial) The reason d'etre of this model is that it not only processes tests inputs but other various other types of data, such as images which we will expore later, for the moment we're just sending text. We could send a mix of data types but for now only text. GPT-4 is refered to as a chat model, chat models are optimied for secnarios where we have a multi-step interactions with the user. This is my current uml for my custom transformer

[5] https://openai.com/index/gpt-4/


Quantum Stack
Physics
MATH/
Philosphy
chem/
AGI
Civics/
Image
computer
Science
Biology/
Ethics
Monte*
carlo
Human
Interface
Telemetry


visual reprocess
abstract reprocess
abstract reprocess
state 1
state 2
state 3
(compare states to reach conclusions & repeat)
send from v1 to v5 then reverse logic & shift forward.
mod of connect
sys state
state change
$\frac{dx}{dt} = (A + \sum_{j=1}^{m} u_j B^{(j)})x + Cu$
for thought visualization
fixed connect
external input



Each node of data must run w/ its own system in mind, whether that be user input system, or a inference based system from re-evaluation.

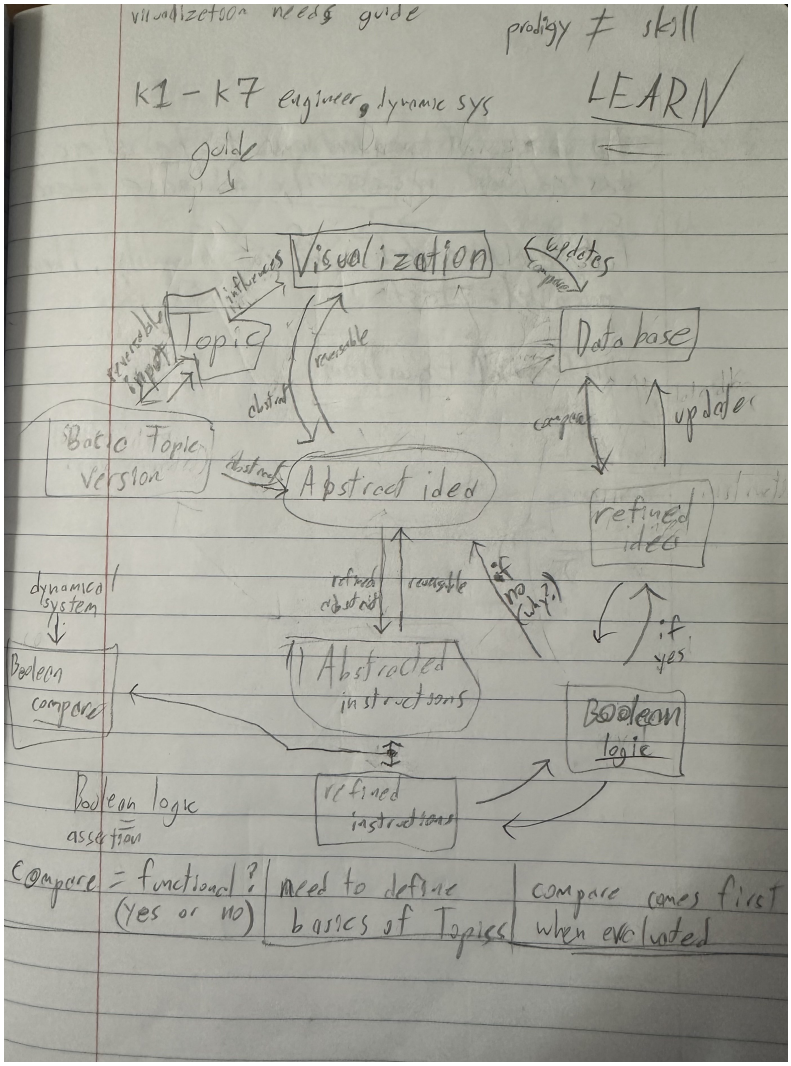
This way they can run independantly, then use the ai itself to call to said databases and visualize a knowledge base and its information attached.

Ask often when approached w/ new info or topic, which topic it originates from.

Scale the system.

Previous system failed mistaking "simplified log" with broad ideas at the base branch of info.

evaluate & re-evaluate options present, view each visualized & abstracted (into idea) conclusion and reverse them at each stage to see if functional.



Back to the tutorial... until i find a place for all my notes

Name	**Institution**	**Qualifications**	**Email**
Jonas Wagner	UT Dallas, UW-Platteville	Mechanical Engineering PhD Student — Engineering Physics and Electrical Engineering Bs	jonas.wagner.2826@gmail.com
Storm Mc.Cauley	UNT	Computer Science Bs student	storm.mccauley@gmail.com
Pavlo Bondarenko	UNT, Odessa Nat'l Polytechnic Univ.	Computer Science Bs student— Computer Engineering Bs	pavlobondarenko@my.unt.edu
Ethan Jensen	UNT, George Fox Univ.	Mathematics PhD Student — Mathematics and Computer Science Bs	ethanjensen@my.unt.edu
Dr. Jacob Hochstetler	UNT, Fidelity Investments	Computer Science and Engineering PhD — VP, Cloud Engineering at Fidelity	Jacob.Hochstetler@unt.edu
Dr. Austen Moss	UNT	Organic Chemistry PhD	austen.moss@unt.edu
Willy Vasquez	UT Austin, MIT	Cryptography PhD Student — Computer Science Ms	wrv@utexas.edu
Hannah Pil	NC State Univ.	Genetics PhD Student	hdpil@ncsu.edu

Table 1: Review Committee Members