Ollscoil Teicneolaíochta an Atlantaigh

Atlantic Technological University

Final Year Project

AI Mentor

Rakib Rana

Bachelor of Software & Electronic Engineering (Honours)

Atlantic Technology University

2024/2025

Declaration

This project is presented in partial fulfilment of the requirements for the degree of Bachelor of Engineering in Software & Electronic Engineering at Atlantic Technology University Galway. This project is my own work, except where otherwise accredited. Where the work of others has been used or incorporated during this project, this is acknowledged and referenced.

Acknowledgements

I greatly appreciate and thank my lecturers, especially Brian O'Shea and Ben Kinsella for helping and guiding me in my project. I also thank my friends and family for providing the resources needed to get my project done and dusted. I believe it has been a great time learning and experiencing something new to everyone.

AI is something that I hope can be used to help students learn and enjoy it while they are at it. I believe motivation is a very important and fortunately I had enough of that to make a very successful product. So, I'm happy with my position thanks to everyone's support and guidance.

Summary

My AI Mentor project is an educational application that helps students learn in subjects like biology and python programming. Especially, with my fine-tuned models that I hand curated and readily made available. The app is also interactive and with flashcards and quizzes to keep everyone engaged.

The interactive system gives extreme detailed feedback, and the progress tracking expands over time. These modes make it fun and exciting to learn and progress further in term of expanding on knowledge.

The project combines React Native for cross-platform frontend functionality with an Express backend hosted on AWS EC2, using MongoDB Atlas for data storage, while the website is hosted on Netlify and running on a custom domain. The star of the show is the custom fine-tuned AI models trained on carefully curated educational datasets derived from Wikipedia, Stack Overflow, and OpenStax materials.

In the future I hope to add voice interaction, file uploads and lots more features while maintaining performance and usability across devices, with potential for much more.

Table of Contents

Summary.....iv

1 Introduction 1

1.1 Project Goals 1

1.2 Scope 1

1.3 Report Overview 1

2 Project Architecture 2

2.1 Architecture diagram 3

2.2 Processing Power Machine 4

2.3 Software Components 4

3 Data Collection and Processing 4

3.1 Data Sources 5

3.1.1 Open Educational Resources (OER) 5

3.1.2 OCR(Optical Character Recognition) 6

3.1.3 Web scraping 6

3.2 Data Extraction 8

3.2.1 Stack Overflow Processing 9

3.3 Hugging Face 11

3.4 Data Filtering 11

3.5 Data Formatting 12

3.5.1 Conversation Structure 14

3.6 Data Balancing 15

4 AI Model Training 16

4.1 AI Models API 16

4.2 Fine-Tuning Preparation 17
4.3 Fine-Tuning Process 19
4.3.1 Hyperparameters 19
4.3.2 Training Monitoring 20
4.4 Model Evaluation 21
5 Backend Development 22
5.1 Server design 22
5.1.1 Secure Communication 23
5.2 Database 24
5.2.1 Schema Design 24
5.3 Dynamic Model Creation 26
5.4 API Endpoints 26
5.4.1 Conversation Management 27
5.4.2 AI Integration 28
5.4.3 Learning Progress Endpoint 28
6 Frontend 29
6.1 Web Application 29
6.1.1 App.js 29
6.1.2 Navigation Component 29
6.1.3 Chat Interface 30
6.1.4 Subject Tutor Selection 31
6.2 Mobile Application 31
6.2.1 Mobile application structure 31
6.2.2 Authentication Flow 32

6.2.3 Learning Features 33

6.2.4 Progress Tracking 34

7 Conclusion 35

8 References 36

9 Code 37

1 Introduction

1.1 Project Goals

My goals…

- Develop an AI-powered tutoring system with specialized subject knowledge
- Create an interactive interface for natural conversational learning
- Implement structured learning tools like flashcards and quizzes
- Build a progress tracking system that adapts to user performance
- Design a cross-platform application with React Native for accessibility

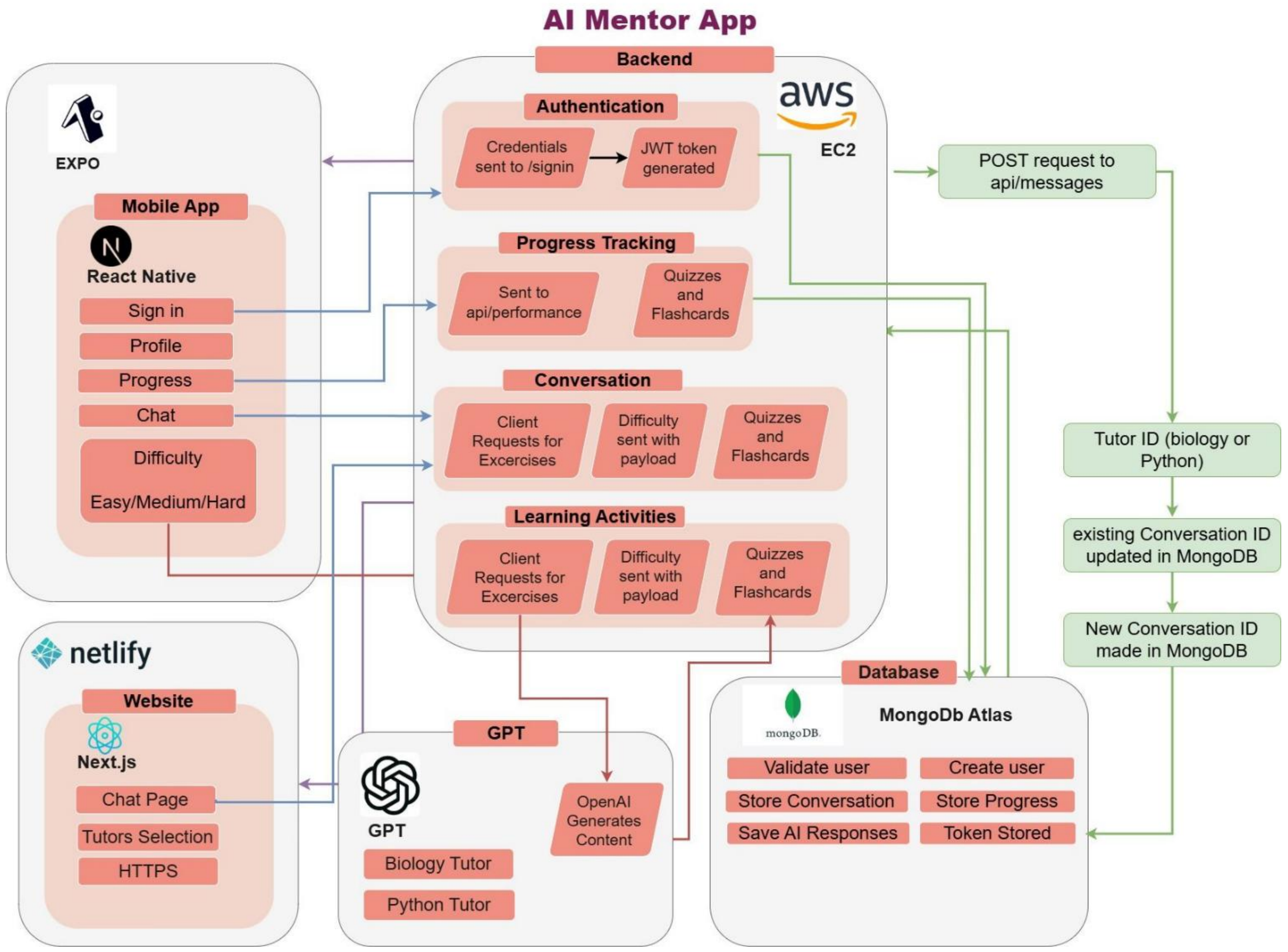
1.2 Scope

The project aims to operate as a complete educational app, providing both tutoring and structured learning activities. The system combines fine-tuned AI models for domain-specific knowledge with a user-friendly interface that allows unified transitions between different learning modes.

1.3 Report Overview

This report documents the complete development process of the AI Mentor system. Including system architecture, component interactions, data collection, processing, scripts, model training methods, mobile and website applications, challenges encountered and their solutions.

2 Project Architecture



2.1 Architecture diagram

- Mobile App built with React Native using Expo
- Website (built with Next.js, hosted on Netlify)
- Backend (running on AWS EC2)

Both my mobile and web application communicate through the same post request, so when creating a conversation EC2 will process the backend in the server.js by making two requests, one to save the user message and one to get the AI response. Both happens whether it's creating a new conversation or sending another message. EC2 stores the user message in MongoDB and when it gets the AI response, it returns it to the client and updates the conversation.

The conversation list is fetched via GET request, and new ones are made via POST through `/api/conversation`

Conversation deletion to through DELETE,

`/api/conversations/${id}?tutor=${tutor}`

All API endpoints are consistent and allows requests to maintain the same chat history.

Both clients try to connect to PORT 443 on EC2 to the same domain, `api.teachmetutor.academy`

On mobile there is also a progress tracking system which monitors learning activities and overall performance on the quizzes and flashcards which is sent to cloud. Sign up and Sign in is done on the app, authenticated with JWT tokens.

2.2 Processing Power Machine

24GB VRAM Graphics card and 32 GB DDR5 used to crunch numbers on big data.

2.3 Software Components

The AI Mentor system utilizes multiple technologies across its components: Data Processing Python 3.10 for data processing scripts, TensorFlow and Hugging Face libraries for data validation

Model Training: OpenAI API for fine-tuning operations, tiktoken for token counting and management, JSON/JSONL formats for training data representation

Backend: Node.js with Express framework, MongoDB Atlas for cloud database, AWS EC2 for deployment, JWT for authentication

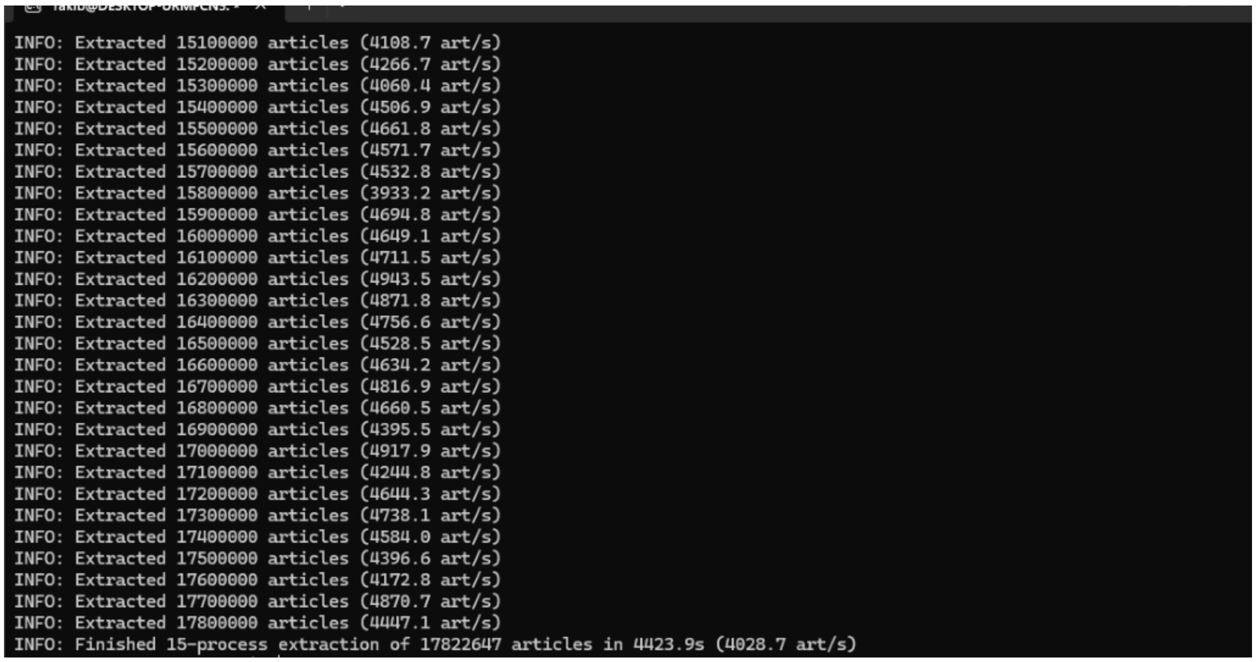
Frontend: React Native with Expo for mobile development, React.js for web interface

3 Data Collection and Processing

This is the most tedious and most important part of the entire project. I would say it’s the heart of the project. To have a good set of fine-tuned models you must first find a good source and then prepare it, so I did that, and I had to go through a lot of difficulties to fine tuning the models and get it working.

3.1 Data Sources

At first, I used Wikipedia dumps with Wiki extractor[1] and tried to get my own database of topics that I liked, it took me from weeks to figure out, until months later I found the combination of Cirrus extractor, lower version python and running the whole thing through Ubuntu (WSL) in Docker was a success.



```
INFO: Extracted 15100000 articles (4108.7 art/s)
INFO: Extracted 15200000 articles (4266.7 art/s)
INFO: Extracted 15300000 articles (4060.4 art/s)
INFO: Extracted 15400000 articles (4506.9 art/s)
INFO: Extracted 15500000 articles (4661.8 art/s)
INFO: Extracted 15600000 articles (4571.7 art/s)
INFO: Extracted 15700000 articles (4532.8 art/s)
INFO: Extracted 15800000 articles (3933.2 art/s)
INFO: Extracted 15900000 articles (4694.8 art/s)
INFO: Extracted 16000000 articles (4649.1 art/s)
INFO: Extracted 16100000 articles (4711.5 art/s)
INFO: Extracted 16200000 articles (4943.5 art/s)
INFO: Extracted 16300000 articles (4871.8 art/s)
INFO: Extracted 16400000 articles (4756.6 art/s)
INFO: Extracted 16500000 articles (4528.5 art/s)
INFO: Extracted 16600000 articles (4634.2 art/s)
INFO: Extracted 16700000 articles (4816.9 art/s)
INFO: Extracted 16800000 articles (4660.5 art/s)
INFO: Extracted 16900000 articles (4395.5 art/s)
INFO: Extracted 17000000 articles (4917.9 art/s)
INFO: Extracted 17100000 articles (4244.8 art/s)
INFO: Extracted 17200000 articles (4644.3 art/s)
INFO: Extracted 17300000 articles (4738.1 art/s)
INFO: Extracted 17400000 articles (4584.0 art/s)
INFO: Extracted 17500000 articles (4396.6 art/s)
INFO: Extracted 17600000 articles (4172.8 art/s)
INFO: Extracted 17700000 articles (4870.7 art/s)
INFO: Extracted 17800000 articles (4447.1 art/s)
INFO: Finished 15-process extraction of 17822647 articles in 4423.9s (4028.7 art/s)
```

Adding a, -Json parameter will help format it into Json which is easy enough to convert to JsonL…

3.1.1 Open Educational Resources (OER)

My initial target was Kaggle[2] as this seemed to be the most popular choice for high quality notes for students. OpenStax[3] also primarily consisted of PDF based content. Other website examples include doabooks , hathitrust, oercommons.

However, all of these have one major problem when it comes to fine tuning for a dataset, they are all mostly comprised of non-text material such as images and icons etc especially for a topic like Biology. Even though PDF are great for human readability, they are much more challenging for machine processing due to the nature of text versus images.

3.1.2 OCR(Optical Character Recognition)

OCR is a very popular method used to extract text from an image for machine processing, however, this comes with random distortions and extremely high processing power, this can be especially costly for just pre-processing, let alone post processing. This will lead to training data contamination and inaccuracies which then model trains on and will pick up these noises and skew the data.

I tried to de-skew the image with filters[4], such as increasing pixel density and sources high quality images, noise removal filters, font choice and better pattern recognition machines (Stronger OCR machine) but even then, some of these were difficult to implement and some even locked behind a paywall.

For example,
‘I love cars’ can be seen in OCR reading as ‘I love cats’

Doing manual review is not feasible on this scale but we are talking about big data, Its more work than necessary.

3.1.3 Web scraping

To address my previous challenge, I decided to do this web scraping, I considered using this as an alternative method for data collection. Web scraping involves automatically extracting information from online such as websites and application. Using this method means I have more control over what I can collect and in almost every way imaginable.

However, web scraping comes with its own set of challenges. For example, attempting to scrape a site with dynamic content might result in incomplete data extraction if the script doesn't execute properly. If JavaScript is blocked partially or fully then progress cannot be made and will lead to incomplete data extraction.

To mitigate these issues, I tried out Selenium[5] which helps with high level JavaScript code on the browser, perfect for web scraping.

```
driver = webdriver.Chrome()

driver.get("https://example.com/articles")

titles = driver.find_elements(By.CLASS_NAME, "article-title")

for title in titles:

    print(title.text)
```

In conclusion, I faced many challenges such as anti-bot measures like captcha’s, dynamic content loading, very slow processing and legal issues.

3.2 Data Extraction

There was a lot of manual code writing such as filtering certain or removing words, keeping the token count limits[6] reasonable or complex pattern recognition to sorting articles, parallel processing and checks, strict message structure all done with the help of Python and its libraries. While the hardest of them all keeping a balance of all of these or else data will get skewed the wrong way, under/over saturated etc.

Docker file was also used to manipulate python version (3.9) as newer versions struggled with wiki extractor.

Anything to improve performance to go through hundreds of gigabytes worth of data on my machine so I didn’t hit a computational limit, but luckily this was never a problem for me given my hardware. I loved seeing how the filtering process evolved, where each iteration of the script got smarter and more complex, finding more precise ways to capture high quality content while filtering out the unwanted noise.

The extraction process used WikiExtractor to convert raw Wikipedia dumps into usable text:

(in a python(py) environment)

```
import os
from wikiextractor.WikiExtractor import main as wiki_extract
config = {
    'input_file': 'enwiki-20240901-pages-articles-
multistream.xml.bz2',
    'output_dir': 'Wiki_extracted_data',
    'json': True,
    'processes': 4
}
```

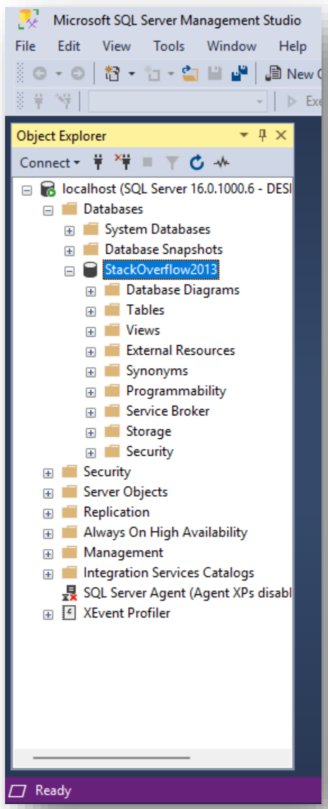
3.2.1 Stack Overflow Processing

Firstly, I downloaded the Stack Overflow XML data dumps from archive.org and from Brent Ozar [7]to my local machine. These files would size would range from 1GB (expanded to 1.5GB) all the way to 47GB (expanded to 180GB) worth of data from StackOverflow.com.

Massive XML files containing posts, users, comments and votes, then filtering these to get a high-quality dataset. This gave me about 20,000 rows/questions to work with. Adjusting to get the perfect balance is key.

SQL Server Management Studio was used to query and extract high quality Python data:

```sql
SELECT
	q.Id AS QuestionId,
	q.Title AS QuestionTitle,
	q.Body AS QuestionBody,
	q.Tags AS Tags,
	a.Id AS AnswerId,
	a.Body AS AnswerBody,
	a.Score AS AnswerScore
FROM Posts q
JOIN Posts a ON q.Id = a.ParentId
WHERE q.PostTypeId = 1 -- Questions
	AND a.PostTypeId = 2 -- Answers
	AND q.Tags LIKE '%python%'
	AND a.Score >= 5
	AND q.Score >= 10
	AND q.AcceptedAnswerId = a.Id
ORDER BY q.Score DESC, a.Score DESC;
```



Here is an SQL query that I ran to filter out the best possible posts from the stack overflow dumps and have enough to eventually fine-tune.

This q.tags here with “python” means to find every post tagged with that word in it.

```
AND q.Tags LIKE '%python%'
```

Also, with an answer rating of more than or equal to 5 and question rating more than or equal to 10.

```
AND a.Score >= 5
AND q.Score >= 10
```

This is a case where only accepted answers are shown.

```
AND q.AcceptedAnswerId = a.Id
```

3.3 Hugging Face

The Hugging Face integration leveraged advanced Python libraries for easy AI model use. It uses transformers for model initialization and a secure API key management. Supported multiple model architectures such as the ones I used Llama-V2 and GPT-2.

This helped facilitate having a ready-made database already fine-tuned and managed by others. This is brilliant as I can move on to making my mobile app or website. However, this was not very flexible, and I had less control over the output since these are already tuned. [8]

3.4 Data Filtering

Raw extracted data underwent rigorous filtering to ensure quality, content quality criteria.

Minimum content length: 500 characters

Maximum content length: 15,000 characters

Subject filtering for Biology content, keyword-based filtering which helps find relevant materials.

```
def is_biology_related(title, text):
    biology_keywords = {
        'biology', 'species', 'cell', 'organism', 'gene', 'evolution',
        'protein', 'bacteria', 'virus', 'animal', 'plant', 'tissue',
        'chromosome', 'dna', 'rna', 'enzyme', 'molecular', 'ecology'
    }
    title_lower = title.lower()
    text_lower = text.lower()
```

Check for Biology keywords in title or text.

```
    if any(keyword in title_lower for keyword in biology_keywords):
        return True
```

Check keyword density in first portion of text.

```
    first_section = text_lower[:1000]
    keyword_count = sum(1 for keyword in biology_keywords if keyword in
first_section)
    return keyword_count >= 3
```

3.5 Data Formatting

Filtered content was transformed into a specific format required for fine-tuning

JSONL Format for each example was structured as a conversation in JSONL format.

```
{
  "messages": [
    {
      "role": "system",
"content": "You are a knowledgeable biology tutor helping users understand
biological concepts."
    },
    {
      "role": "user",
      "content": "Please explain the concept of photosynthesis in biology”
    },
    {
      "role": "assistant",
      "content": "Photosynthesis is the process by which green plants, algae, and some bacteria convert light energy into chemical energy. “
    }
  ]
}
```

3.5.1 Conversation Structure

The system message was tailored to the specific subject domain to provide appropriate context for the model:

```
def create_training_example(title, text):
    system_message = (
        "You are a knowledgeable biology tutor. Provide accurate, 
educational, "
        "and engaging information about biological concepts. Break down 
complex "
        "topics into understandable explanations."
    )
```

Clean and format the text.

```
    text = text.replace('\n\n', ' ').strip()
    if len(text) > 2000:
        text = text[:2000] + '...'

    messages = [
        {"role": "system", "content": system_message},
        {"role": "user", "content": f"Please explain the concept of {title} 
in biology. What are its key characteristics and significance?"},
        {"role": "assistant", "content": text}
    ]

    return {"messages": messages}
```

3.6 Data Balancing

When using the Wikipedia data, from my research I strongly recommend around 5000+ articles per topic as this gave the best bang for your buck, both in term of money and quality. 2000+ articles per subject is roughly enough detail for that subject for fine-tuning but you won't get the greatest results and anything below that is not recommended (by me). Below is an example of when I tried to extract multiple topics at once.

- Biology: 1,984 articles
- Computer Science: 657 articles
- Physics: 2,575 articles
- Mathematics: 693 articles
- Chemistry: 2,517 articles
- History: 10,000 articles
- Balancing Strategy:

To create balanced datasets, a combination of techniques was used such as random sampling for overrepresented topics, code changes for underrepresented topics and targeted extraction for specialized topics.

Through research I found that the best distribution of data is about

- 70 common knowledge
- 20 uncommon topics
- 10 advanced topics

This gives the user the best experience to learn from and progress further, anything too much of advanced topics for example will lead to extreme difficulty bias.

The final balanced datasets contained 3,500 examples per subject, ensuring models wouldn't develop biases toward subtopics.

4 AI Model Training

4.1 AI Models API

Selecting the right base model was important as each have their own requirements and needs.

The ideal model for me would be a balance of cheap and reliable. Strong baseline knowledge of educational subjects. Suitable context window for educational explanations. Reasonable fine-tuning costs. Manageable inference latency for real-time conversations

Model Comparison below

Model	Knowledge	Context Size	Fine-tuning Cost	Cost (1M Token)
GPT 3.5	Medium	4k	Medium	€0.47
GPT 4	Best	8k	Expensive	€27
Llama 2	Low	1k	Low	€0.1

GPT 3.5 is very balanced and is perfect for my use case in terms of cost and efficiency.

4.2 Fine-Tuning Preparation

Validating the data extracted is just as important if not even more important than data preparation, as this is the last stage before fine tuning. Whatever you have decided to make of your dataset will now be coming to fruition.

Now with extremely high-quality topic specific data I used the OpenAI API to fine tune all my models. I found great resources online to help me with all these details as information on fine-tuning weren't public when I started this project. OpenAI Cookbook [11] has helped with this information. The first step would be to validate conversation format and total token count.

```
def validate_conversation(conversation):
    try:
        if 'messages' not in conversation:
            return False, "missing_messages"
        messages = conversation['messages']

        if not all(isinstance(m, dict) and 'role' in m and 'content' in m for
m in messages):
            return False, "invalid_message_structure"

        for message in messages:
            content_length = len(message['content'])
            if not (MIN_CONTENT_LENGTH <= content_length <=
MAX_CONTENT_LENGTH):
                return False, f"content_length_{content_length}"

        total_tokens = sum(len(encoding.encode(m['content'])) for m in
messages)
        if not (MIN_TOKENS <= total_tokens <= MAX_TOKENS):
            return False, f"token_count_{total_tokens}"
        return True, "valid"
    except Exception as e:
        return False, f"error_{str(e)}"
```

Token usage was estimated to predict training costs: €0.05 per 1K tokens for GPT-3.5-Turbo. Count tokens in a JSONL file to estimate fine-tuning costs. This is good for understanding how much your dataset will cost to train.

```
def count_tokens_in_jsonl_file(file_path):
    encoding = tiktoken.encoding_for_model("gpt-3.5-turbo")
    total_tokens = 0

    with open(file_path, 'r', encoding='utf-8') as file:
        for line in file:
            conversation = json.loads(line)
            for message in conversation['messages']:
                total_tokens += len(encoding.encode(message['content']))


    cost_per_1k_tokens = 0.008
    estimated_cost = (total_tokens / 1000) * cost_per_1k_tokens

    return total_tokens, estimated_cost
```

4.3 Fine-Tuning Process

4.3.1 Hyperparameters

Fine-tuning parameters were selected based on the OpenAI documentation.

```
Python_training_config = {
    'model': 'gpt-3.5-turbo',
    'training_file': 'biology_training_data.jsonl',
    'hyperparameters': {
        'learning_rate_multiplier': 0.1,
        'batch_size': 4,
        'n_epochs': 3
    }
}
```

4.3.2 Training Monitoring

The training process was monitored through the OpenAI dashboard and API:

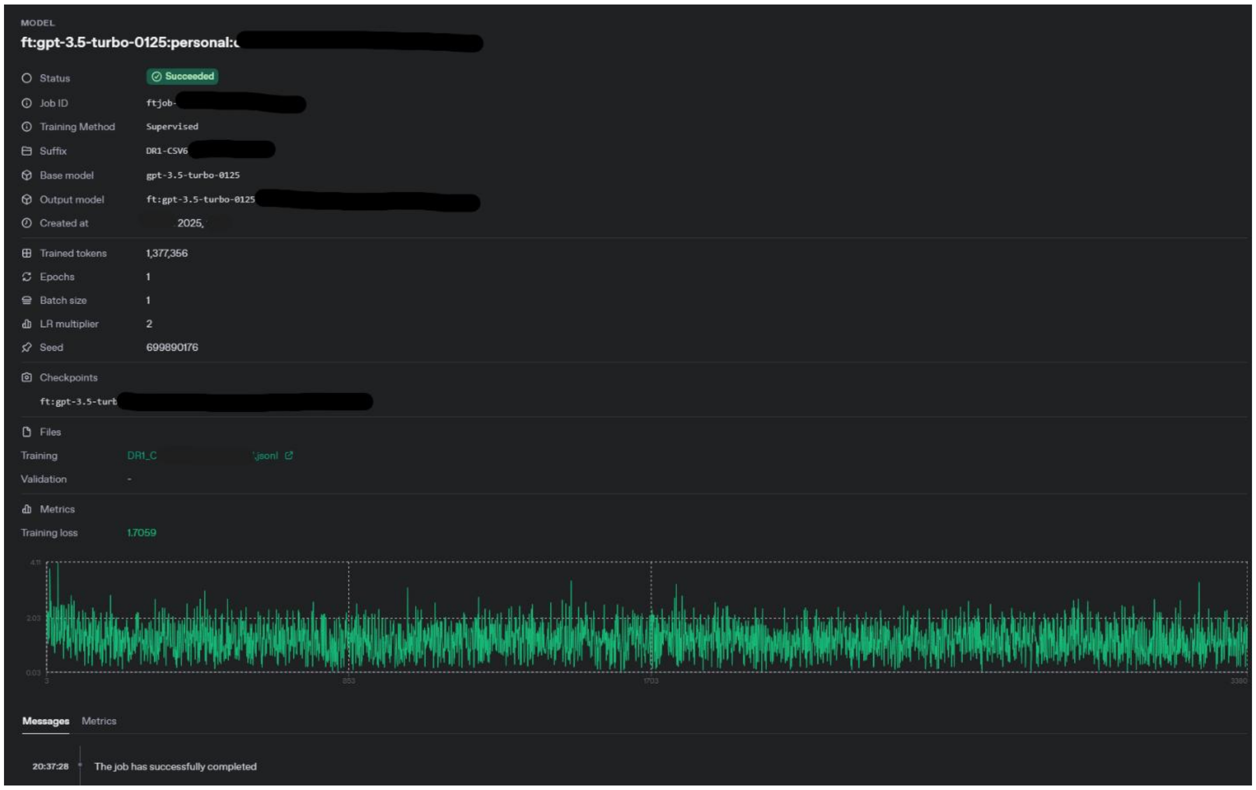
```
def check_job_status(job_id):
    job = openai.FineTuningJob.retrieve(job_id)

    print(f"Job status: {job.status}")
    print(f"Created at: {job.created_at}")
    print(f"Updated at: {job.updated_at}")

    if job.status == "succeeded":
        print(f"Fine-tuned model: {job.fine_tuned_model}")
        return job.fine_tuned_model

 return None
```

Figure 4.3.1 shows my training loss curves from the fine-tuning process.



MODEL
ft:gpt-3.5-turbo-0125:personal:[redacted]

Status Succeeded
Job ID ftjob-[redacted]
Training Method Supervised
Suffix DR1-CSV6[redacted]
Base model gpt-3.5-turbo-0125
Output model ft:gpt-3.5-turbo-0125[redacted]
Created at 2025,

Trained tokens 1,377,356
Epochs 1
Batch size 1
LR multiplier 2
Seed 699890176

Checkpoints
ft:gpt-3.5-turb[redacted]

Files
Training DR1_C '.jsonl
Validation -

Metrics
Training loss 1.7059

20:37:28 The job has successfully completed

4.4 Model Evaluation

First, I tracked how well the models were learning during training by watching their loss values decrease over time as this was a way to tell they were improving with each training cycle.

I then measured how efficient the models were with their token usage - basically checking if they provided short but good responses rather than a whole lot of nothing.

Finally, I conducted comparison testing by putting my fine-tuned models against both the standard GPT-3.5 model (without any specialized training) and other Chatbots AI tools. The results clearly showed that my custom-trained models performed significantly better in delivering accurate, subject-specific knowledge in both Biology and Python.

5 Backend Development

5.1 Server design

The backend environment uses Node.js with an express framework.

From server.js

```
const express = require("express");

const mongoose = require("mongoose");
const cors = require("cors");
const fetch = require("node-fetch");
const bcrypt = require("bcryptjs");
const jwt = require("jsonwebtoken");
```

Environment configuration.

```
require("dotenv").config();
const PORT = process.env.PORT || 3000;
const HTTPS_PORT = process.env.HTTPS_PORT || 443;
```

Initialize Express application.

```
const app = express();
app.use(express.json());
app.use(cors({
  origin: ['http://localhost:3000', 'https://tutor.com,
           'https://www.tutor.com, 'https://tutor.netlify.app'],
  credentials: true
}))
```

The backend follows a modular design with specialized services such as authentication, learning and ai service for login, progress tracking and API inferences.

5.1.1 Secure Communication

The server implements both HTTP and HTTPS protocols, with automatic redirection to secure connections when SSL certificates are available.

SSL Configuration

```
let sslOptions;
try {
  sslOptions = {
    key: fs.readFileSync('/etc/letsencrypt/live/tutor.com/privkey.pem'),
    cert: fs.readFileSync('/etc/letsencrypt/live/tutor.ie/fullchain.pem'),
    ca: fs.readFileSync('/etc/letsencrypt/live/tutor.org/chain.pem')
  }
  console.log("SSL certificates loaded successfully");
} catch (err) {
  console.warn("SSL certificates not found or couldn't be loaded:",
err.message);
  sslOptions = null;
}
```

HTTP to HTTPS redirect middleware.

```
if (sslOptions) {
  app.use((req, res, next) => {
    if (!req.secure && req.get('x-forwarded-proto') !== 'https') {
      const host = req.headers.host || tutor.com;
      return res.redirect(`https://${host}${req.url}`);
    }
    next()
  })
}
```

5.2 Database

The MongoDB database architecture is designed to support the educational focus of the application while maintaining flexibility for future expansion. Several core schemas define the data structure.

5.2.1 Schema Design

User Schema:

```
const userSchema = new mongoose.Schema({
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true },
  fullName: { type: String, required: true, default: '' },
  createdAt: { type: Date, default: Date.now }
})
```

Conversation Schema:

```
const conversationSchema = new mongoose.Schema({
  title: String,
  messages: [{
    sender: String,
    text: String,
    attachments: [String],
    timestamp: { type: Date, default: Date.now }
  }],
  model: String,
  createdAt: { type: Date, default: Date.now }
})
```

Performance Schema:

```
const performanceSchema = new mongoose.Schema({
  userId: { type: String, required: true },
  tutor: { type: String, required: true },
  topic: { type: String, required: true },
  subtopic: { type: String, default: 'general' },
  activityType: { type: String, enum: ['flashcard', 'quiz'], required: true
},
  cards: [{
    cardId: String,
    question: String,
    answer: String,
    subtopic: { type: String, default: 'general' },
    attempts: { type: Number, default: 0 },
    correctAttempts: { type: Number, default: 0 },
    lastAttempt: { type: Date, default: Date.now },
    difficulty: { type: Number, default: 3 }
  }],
  sessions: [{
    date: { type: Date, default: Date.now },
    subtopic: { type: String, default: 'general' },
    cardsStudied: Number,
    correctAnswers: Number,
    timeSpent: Number
  }],
  createdAt: { type: Date, default: Date.now },
  updatedAt: { type: Date, default: Date.now }
})
```

5.3 Dynamic Model Creation

The system dynamically creates models based on tutors and users to maintain subject separation.

```
function getConversationModel(tutor) {
  const name = "Conversation_" + tutor;
  return mongoose.models[name] ||
         mongoose.model(name, conversationSchema, "conversations_" + tutor);
}

function getPerformanceModel(userId) {
  const name = "Performance_" + userId;
  return mongoose.models[name] ||
         mongoose.model(name, performanceSchema, "performances_" + userId);
}
```

5.4 API Endpoints

The backend exposes several RESTful API endpoints to support client side functions like authentication endpoints and sign up and sign in.

```
app.post("/signup", async (req, res) => {
  const { email, pass, fullName } = req.body;
  console.log("Signup request:", { email, fullName });

app.post("/signin", async (req, res) => {
  const { email, pass } = req.body;
```

5.4.1 Conversation Management

For each of my tutor, you will be able to get the conversation, create a new one or continue from where you left off.

Example of creating a new conversation.

```
app.get("/api/conversations", async (req, res) => {
```

Example of continuing a message.

```
app.post("/api/messages", async (req, res) => {
```

Example of deleting a new conversation.

```
app.delete("/api/conversations/:id", async (req, res) => {
```

Example of fetching a conversation.

```
app.get("/api/conversations", async (req, res) => {
  try {
    const tutor = req.query.tutor;
    if (!tutor) return res.status(400).json({ error: "Tutor query parameter
is required" });
    const Conv = getConversationModel(tutor);
    const list = await Conv.find().sort({ createdAt: -1 });
    res.json(list);
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: "Server error" });
  }
})
```

5.4.2 AI Integration

OpenAI Endpoint.

```
app.post("/api/openai", async (req, res) => {
```

Tutor specific prompting.

```
  const tutorPrompts = {
    biology: "You are a Biology tutor specialising in genetics, ecology,
    physiology.",
    python: "You are a Python programming tutor helping with syntax and
    debugging.",
    maths: "You are a Maths tutor covering algebra to calculus.",
    english: "You are an English tutor focusing on grammar and literature."
  }
```

5.4.3 Learning Progress Endpoint

Get performance data

```
    app.get("/api/performance", async (req, res) => {
      try {
        const { userId, tutor, topic, subtopic, activityType } = req.query;
```

Update performance data

```
    app.post("/api/performance", async (req, res) => {
      try {
        const { userId, tutor, topic, subtopic, activityType, sessionData,
    cardsData } = req.body;
```

Get subtopic progress

```
    app.get("/api/progress/subtopics", async (req, res) => { try { const {
    userId, tutor } = req.query;
```

6 Frontend

6.1 Web Application

The web application is built with React.js, providing a responsive interface for users to interact with AI tutors and access learning materials.

6.1.1 App.js

The application is organized into components and pages.

```
<Routes>
  <Route path="/" element={<Home />} />
  <Route path="/chat" element={<Chat />} />
  <Route path="/about" element={<About />} />
  <Route path="/subject-tutor" element={<SubjectTutor />} />
</Routes>
```

6.1.2 Navigation Component

Navigation component used to move between headers. Here’s some example selections.

```
<li>
  <Link to="/">Home</Link>
</li>
<li>
  <Link to="/chat">Chat</Link>
</li>
<li>
  <Link to="/about">About</Link>
</li>
<li>
  <Link to="/subject-tutor">Subject Tutor</Link>
</li>
```

6.1.3 Chat Interface

The chat interface allows conversations. Below are some state variables to track which tutors are selected, what model is being used and listing the entire conversation.

```
function Chat() {
  const location = useLocation();
  const [tutor, setTutor] = useState(initialTutor);
  const [selectedModel, setSelectedModel] = useState(initialModel);
  const [conversations, setConversations] = useState([]);
  const [activeConversationId, setActiveConversationId] =
useState(initialConversationId);
  const [userInput, setUserInput] = useState("");
  const [isLoading, setIsLoading] = useState(false);
```

6.1.4 Subject Tutor Selection

The application allows users to select specific subject tutors

```
const handleSelectSubject = async (subject) => {
  const                   response                  =                  await
fetch('https://api.teachmetutor.academy/api/conversations', {
    method: 'POST',
    body: JSON.stringify({
      title: subject.name,
      model: subject.model,
      tutor: subject.id
    })
  })
}
```

This allows making a new conversation when a tutor is selected and redirects too.

6.2 Mobile Application

The mobile application is built using React Native with Expo, allowing for deployment on both iOS and Android platforms.

6.2.1 Mobile application structure

The mobile app uses React Navigation for screen management.

```
export default function App() {
  return (
    <UserProvider>
      <AuthProvider>
        <AppNavigator />
        <StatusBar style="light" />
      </AuthProvider>
    </UserProvider>
  )
```

6.2.2 Authentication Flow

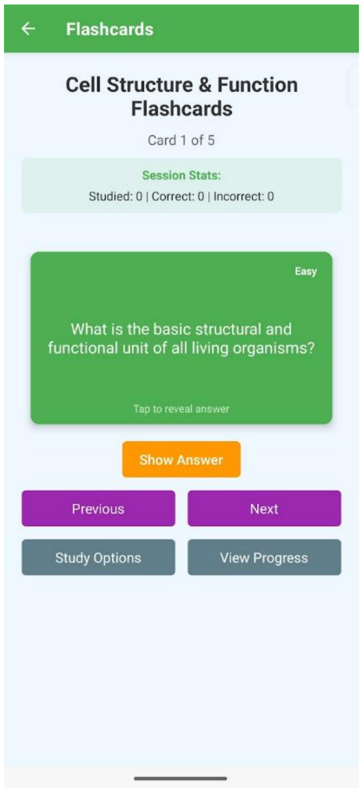
The mobile app includes a complete authentication system.

```
const handleSignIn = async () => {
  if (!validateForm()) return;
  try {
    await signIn(email, password);
  } catch (error) {
  }
}
```

This validates user input and attempts to sign in, with navigation happening automatically based on authentication state.

6.2.3 Learning Features

The mobile app includes specialized learning features like flashcards and quizzes.


Flashcards

Cell Structure & Function Flashcards

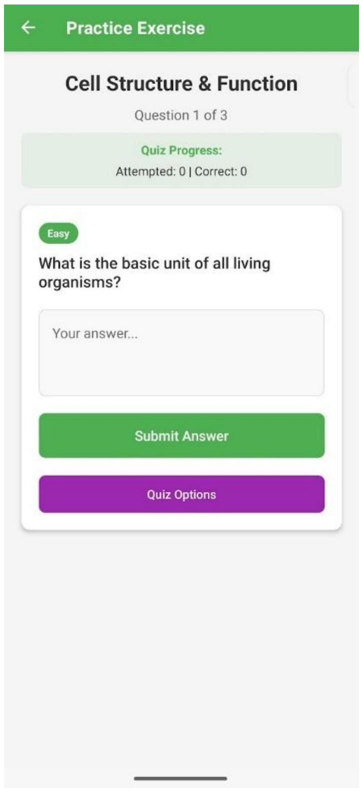
Card 1 of 5

Session Stats:
Studied: 0 | Correct: 0 | Incorrect: 0

Easy

What is the basic structural and functional unit of all living organisms?

Tap to reveal answer


Practice Exercise

Cell Structure & Function

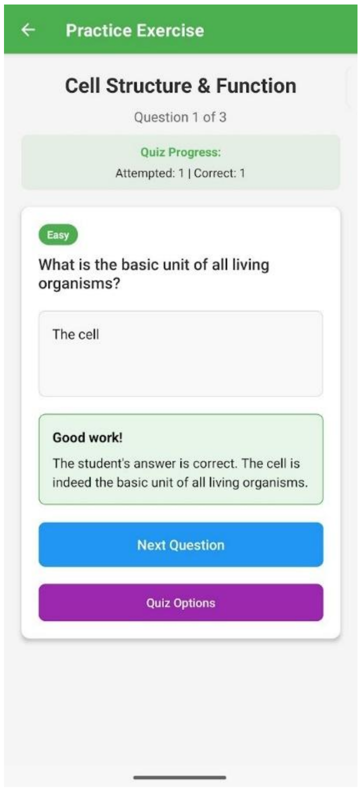
Question 1 of 3

Quiz Progress:
Attempted: 0 | Correct: 0

Easy

What is the basic unit of all living organisms?

Your answer...


Practice Exercise

Cell Structure & Function

Question 1 of 3

Quiz Progress:
Attempted: 1 | Correct: 1

Easy

What is the basic unit of all living organisms?

The cell

Good work!
The student's answer is correct. The cell is indeed the basic unit of all living organisms.

6.2.4 Progress Tracking

The app includes a progress tracking feature so you can easily see how proficient you are at a topic. Each time a student completes a flashcard or quiz. The app records which topics they studied and how well they did. Data such as the number of questions attempted right and wrong, time spent and chosen difficulty level.

All data is stored on MongoDB under that user's profile. You can get an overview of all topics in one place and easily find out what needs to be practiced more.

```
const getMasteryText = (level) => {
  switch (level) {
    case 0: return 'Not Started';
    case 1: return 'Beginner';
    case 2: return 'Developing';
    case 3: return 'Competent';
    case 4: return 'Proficient';
    case 5: return 'Expert';
    default: return 'Unknown';
  }
}
```

This function converts numerical mastery levels into text for display in the progress interface.

The progress is dynamically visualised using a progress bar that visually represents the users learning progress as a percentage.

```
<View style={styles.progressBarContainer}>
  <View style={[styles.progressBar, { width: `${progress.overall}%` }]} />
  <Text style={styles.progressText}>{progress.overall}%</Text>
</View>
```

7 Conclusion

I believe that this is a great resource to learn and challenge yourself to educate and master. Motivation is a real killer and I'm sure making learning fun is extremely important, this application can do just that. A tutor right in your pocket ready to track your progress all at your own pace. By making it simple and clear on what the needs to be done this app can be very valuable as it may even show a weakness that you're not aware of.

From all my productive research and slow data nitpicking over the months, I have gained a lot of experience on fine-tuning models, data crunching, filtering and have been successful building a whole application. Thanks for Reading!

8 References

[1] AWS (Amazon Web Services). "Amazon EC2." https://aws.amazon.com/ec2/

[2] MongoDB Inc. "MongoDB Atlas." https://www.mongodb.com/cloud/atlas

[3] Attardi, G. (2015). "WikiExtractor." https://github.com/attardi/wikiextractor

[4] OpenAI. "Managing tokens in the OpenAI API." https://platform.openai.com/docs/guides/fine-tuning/managing-tokens

[5] Kaggle Inc. "Kaggle Datasets." https://www.kaggle.com/datasets

[6] OpenStax. "OpenStax Biology 2e." https://openstax.org/details/books/biology-2e

[7] OpenCV Team. "Image Processing in OpenCV." https://docs.opencv.org/master/d2/d96/tutorial_py_table_of_contents_imgproc.html

[8] SeleniumHQ. "Selenium WebDriver." https://www.selenium.dev/documentation/en/webdriver/

[9] Hugging Face Inc. "Transformers." https://huggingface.co/docs/transformers/index

[10] Ozar, B. "Stack Overflow Database Download." https://www.brentozar.com/archive/2015/10/how-to-download-the-stack-overflow-database-via-bittorrent/

[11] OpenAI. "OpenAI Cookbook." https://github.com/openai/openai-cookbook

[12] OpenAI. "Chat completion." https://platform.openai.com/docs/guides/text-generation/chat-completions-api

[13] React Native Community. "React Native." https://reactnative.dev/docs/getting-started

[14] Facebook Inc. "React." https://reactjs.org/docs/getting-started.html

[15] MongoDB Inc. "Mongoose." https://mongoosejs.com/docs/

[16] Express.js. "Express." https://expressjs.com/en/guide/routing.html

[17] ChatGPT "OpenAI" https://chatgpt.com/

9 Code

Including all other GitHub links.

https://github.com/RakibR7/AI_Mentor_FYP

All code can be found at the following GitHub.

https://github.com/RakibR7/AI_Mentor_Database

https://github.com/RakibR7/Mobile-App

https://github.com/RakibR7/AI_Mentor_Backend

https://github.com/RakibR7/AI_Mentor_Website