Technical Report Advanced AI

Introduction

When people only have a few ingredients left at home, they often struggle to come up with a recipe and don’t want to go to the store or order takeout. This project aims to solve that problem by providing an AI-powered recipe generator that takes a list of available ingredients and produces complete, creative recipes.

As someone who lacks creativity in the kitchen, this idea was personally appealing. I often find myself unsure of what to cook, even when I have several items in my fridge. This project offers a solution that could help both me and others in similar situations.

The chosen approach was to take the Llama-3.2-1B-Instruct model and fine tune it to understand more about ingredients and recipes. Then place this newly trained model inside a desktop GUI application built with Tkinter. This application serves as a lightweight recipe-generation tool.

Note: this application was built and made on/for MacOS. The gui.py script runs on any OS of course, but I have used Platypus (a MacOS app designed to bundle files to run harmoniously, becoming a desktop app) to bundle the script with its used models, virtual environment and launcher. This way, it becomes an actual desktop app, causing for any non-technical user to be able and use the application with ease.

GitHub Repository:  
<https://github.com/ED-ISA-243/AdvancedAI_IndividualProject>

# Data

# I started by downloading a dataset from Kaggle. This massive dataset consisted of the following headers:

* Id (primary key)
* Title (name of the recipe)
* Ingredients (full ingredient list with amounts)
* Description (how to make the recipe)
* Link (URL where recipe was found)
* Source (name of URL website name)
* Ner (Named Entity Recognition, purely ingredient names without amounts)

I then took several steps in cleaning this dataset, as you can see in the data\_setup.ipynb file. After taking several steps, I finally landed the train.jsonl and val.jsonl files, being the correct data needed to train the model.  
Train.jsonl contains of 199863 rows, and val.jsonl exists of 4079 rows. After already having trained and evaluated the model, I remembered the norm is rather to split the data 80-20, which I didn’t do. However, it does not matter since I have only used 5% of the train.jsonl data, since it took too long to train on all data.

# Model & Methods

# At first, I tried fine-tuning Llama-3.2-3B-Instruct, but training the full dataset took more than 48 hours, which made it unpractical to run without major limitations. I then switched to the lighter 1B model, which reduced training time to ~12 hours for the full dataset and just 10–15 minutes for 5% of the data . This 5% still contained around 10k rows, which is still valid for model training. After training followed evaluation on val.jsonl.

# The fine-tuning was done with supervised training on a CLM (causal language modeling task), where the model learns to predict the next token in a recipe. To make this efficient, I used LoRA adapters, which add small trainable matrices to the model while keeping the original weights frozen, so far fewer parameters need updating. Data was tokenized into fixed-length sequences with padding and EOS markers, and training used AdamW with a cosine learning rate schedule and warmup to ensure stability.

# After training, only the LoRA adapter is saved, producing a lightweight artifact that can be merged with the base model for inference. This setup allowed me to train a capable recipe generator while avoiding the impractical runtimes of the larger model.

# **Results & Evaluation**

Training results:

* Training loss decreased from 1.7 to 1.0 🡪 model learned stably
* Gradient norms stayed under 1.0
* Mean token accuracy climbed from 65% to 75%
* Learning rate followed a cosine schedule, starting at 1e-4 and smoothly decreasing  
  stable and efficient training progression
* Final training loss ~1.1 → perplexity ≈ 3.0, meaning the model is confident when predicting tokens (tokens being basic text units the model reads/predicts)
* Validation loss ~1.06 with perplexity ~2.9 → consistent with training, no signs of overfitting

Eventually the LoRA (Low-Rank Adaptation) adapter was successfully saved and ready to merge with the base model for inference. The adapter by itself is not a full model, needing the default Llama-3.2-1B-Instruct as a base to work.

If I were to train the model even further, it is safe to assume the results would probably improve even further.

Recipe results:

* Model is very fast and lightweight
* Understands input clearly and knows what to do with it
* Generated properly structured recipe with used ingredients list and descriptions
* Model can’t handle list with too many ingredients
* Model refused to give multiple unique recipes, had to reduce to 1 recipe max.
* Model often can’t provide 5 steps, so I added a filler.

# Contributions

I have worked on this project alone, with using GenAI as my copilot. The GenAI e-helped me discover and decide on:

* Ideas for used training methods
* choice of model (Llama-3.2-1B-Instruct)
* how to make the application non-technical user friendly
* jsonl
* how to approach code
* …

# Challenges & Future work

This project was very interesting, seeing how it evolved from managing the raw data to a working model generating recipes, which I fine-tuned on that data. It was very challenging for me, since I had 0 knowledge on how to fine-tune a LLM at first, since we’ve mainly learned how to partake in regression, classification, gradient boosts, … code wise.

In the future, I would focus on training the model for a full 12 hours, splitting this into sections of around 2 hours each with evaluations in between to monitor progress. I would also work on improving the quality and clarity of the generated instructions.