Recipe Generation Using Small Language Models

Student Name: Kanak Yadav
Roll Number: 2022611

BTP report submitted in partial fulfillment of the requirements
for the Degree of B.Tech. in Computer Science and Applied Mathematics

on April 23, 2025

BTP Track: BTP Track

Dr. Ganesh Bagler

**Indraprastha Institute of Information Technology
New Delhi**

Student's Declaration

I hereby declare that the work presented in the report entitled **Recipe Generation Using Small Language Models** submitted by me for the partial fulfillment of the requirements for the degree of *Bachelor of Technology* in *Computer Science & Applied Mathematics* at Indraprastha Institute of Information Technology, Delhi, is an authentic record of my work carried out under guidance of **Dr. Ganesh Bagler**. Due acknowledgements have been given in the report to all material used. This work has not been submitted anywhere else for the reward of any other degree.

.... **Place & Date: IIIT-D 23-Apr-2025**
Kanak Yadav

Certificate

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

.... **Place & Date: IIIT-D 23-Apr-2025**
Dr. Ganesh Bagler

Abstract

The automatic generation of cooking recipes has become more popular in recent years, thanks to the large number of recipes available online. In this project, we built a system that can both create and check recipes. The main goal of the system is to generate recipe instructions based on given recipe title and a list of ingredients. In this project, we fine-tune small and efficient language models such as T5 Small, DistilBERT, and DistilRoBERTa to generate cooking instructions using a recipe title and a list of ingredients. These models are trained on a collection of recipes to understand how instructions are usually written. Because these models are small, they run faster and are better suited for devices with limited memory and processing power. Our results show that when fine-tuned properly, even small models can create clear and useful recipe instructions. This makes AI-powered recipe creation easier to use in real life. We are also comparing the performance of these small models to better understand their strengths and see which one works best for recipe Generation.

Acknowledgments

I would like to express my sincere gratitude to Professor **Dr.Ganesh Bagler** for his guidance and support throughout this project. His insights into computational gastronomy and the intersection of data science with food studies have been truly inspiring. The opportunity to explore recipe generation using language models under his mentorship has been both enriching and intellectually rewarding. This work would not have been possible without his encouragement and valuable feedback at every stage.

Contents

Acknowledgments i

1 Introduction **1**

1.1 Problem Statement and Motivation 2

1.1.1 Problem Statement 2

1.1.2 Motivation 2

2 Literature Review **3**

3 Methodology **5**

3.0.1 1. Data Exploration & Preprocessing 5

3.0.2 2. Model Setup & Prompting 5

3.0.3 3. Training & Fine-Tuning 6

3.0.4 4. Evaluation 7

4 Implementation **8**

4.0.1 1. Exploratory Data Analysis & Preprocessing 8

4.0.2 2. Modular Script Structure 9

5 Results and Analysis **12**

5.0.1 DistilBERT: Fine-Tuning Performance Analysis 12

5.0.2 DistilRoBERTa: Fine-Tuning and Evaluation Analysis 13

5.0.3 T5-Small: Training Dynamics and Generation Quality 15

5.0.4 Comparison of Model Performance 16

6 Conclusion and Future Work **18**

6.0.1 Conclusion 18

6.0.2 Future Work 18

Chapter 1

Introduction

The automatic generation of cooking recipes instrution presents an innovative and highly practical solution for enhancing user experience in culinary applications. Traditional recipe retrieval systems, such as those found on Allrecipes or Yummly, often require users to navigate complex search interfaces, which may be overwhelming for beginners. These systems also typically lack creative flexibility, offering only pre-existing combinations rather than novel culinary possibilities. To address these limitations, recipe generation systems have emerged, aiming to automatically produce detailed cooking instructions or ingredient lists based on user-specified inputs.

Recent advancements in Natural Language Processing (NLP) have demonstrated the potential of deep learning models, particularly large-scale transformer-based architectures, in generating coherent and contextually rich text. Among these, OpenAI's GPT-2 has achieved state-of-the-art performance in various text generation tasks by effectively modeling long-range dependencies and linguistic structures. However, the focus has predominantly been on large models, and their applicability in domain-specific tasks such as culinary instruction generation remains under-explored.

This project introduces an approach for recipe generation using small language models, offering a lightweight yet effective alternative for both instruction generation. Unlike large models that demand significant computational resources, small language models can deliver efficient performance while maintaining practical usability in resource-constrained environments. The proposed system leverages these models to generate recipes from user-defined titles and ingredients, enabling both functional cooking assistance and creative exploration in culinary arts.

1.1 Problem Statement and Motivation

1.1.1 Problem Statement

Automatic recipe generation systems must translate a recipe title and ingredient list into coherent, step-by-step cooking instructions—a task increasingly informed by vast online recipe repositories. While large pre-trained models like GPT-2 and GPT-3 excel at fluency, their size (hundreds of millions to billions of parameters) leads to high inference latency and makes them impractical for deployment on edge devices or in low-resource environments. Moreover, unconstrained generation often omits required ingredients or misorders critical cooking steps, undermining recipe validity and user trust. There is thus a pressing need for models that (1) deliver high-quality, contextually accurate recipe instructions, (2) maintain ingredient–instruction consistency, and (3) operate efficiently on limited hardware.

1.1.2 Motivation

Growing Demand for Culinary Automation Consumers and professionals alike seek tools that can rapidly synthesize novel recipes tailored to dietary preferences, available ingredients, or nutritional goals. Automated recipe generation empowers personalized meal planning, reduces decision fatigue, and enhances culinary creativity without requiring expert knowledge.

Limitations of Large Language Models Although models such as GPT-2 augmented with Monte Carlo Tree Search (RecipeMC) improve recipe plausibility, they remain computationally intensive and unsuitable for on-device applications. Instruction-tuned models like Orca-2 demonstrate strong zero-shot performance but at the cost of even greater parameter counts, hindering responsive, decentralized inference.

Advantages of Small, Distilled Models Models distilled via knowledge-distillation (e.g., DistilBERT) or designed as compact encoder–decoder architectures (e.g., T5-Small) can achieve comparable language understanding on numerous NLP tasks while reducing model size by up to 60% and inference time by over 40%. Fine-tuning these small models specifically on recipe datasets leverages their efficiency, enabling practical deployment in cooking assistants, mobile apps, and IoT kitchen devices.

Domain-Specific Evaluation Standard metrics (BLEU, ROUGE) only partially capture recipe quality; thus, we integrate domain-specific evaluations such as ingredient coverage checks and logical sequencing consistency, drawing on benchmarks like Recipe1M+ and content-planning frameworks to ensure generated instructions are both accurate and actionable.

Chapter 2

Literature Review

The task of automatic recipe generation has gained considerable momentum with the rapid development of natural language processing (NLP) technologies and the availability of large online culinary datasets. Early efforts in this domain primarily utilized large pre-trained models such as GPT-2, which, when fine-tuned on recipe datasets, were capable of producing detailed, coherent instructions from simple prompts like recipe titles and ingredient lists. These models demonstrated superior performance over older character-level models such as LSTMs in terms of fluency and diversity of output. Public implementations showcased their ability to generate structured recipe components, including ingredients and stepwise instructions. However, the high computational requirements of large models like GPT-2 Medium and Large (ranging from 350M to 1.5B parameters) posed practical challenges, particularly for deployment in resource-constrained environments.

To address these limitations, recent research has shifted focus towards compact and efficient language models. Models such as DistilBERT (66M parameters), DistilRoBERTa (82M parameters), and T5 Small (60M parameters) have been explored for their potential in recipe generation tasks. These models are derived through techniques such as knowledge distillation, allowing them to retain a substantial portion of their larger counterparts' capabilities while significantly reducing inference time and memory usage. For instance, DistilBERT and DistilRoBERTa have been fine-tuned for tasks like ingredient recognition and masked token prediction in cooking contexts, demonstrating competence in generating relevant text completions. T5 Small, which frames all NLP problems as text-to-text tasks, has been particularly effective in generating instructions from given ingredients and titles due to its flexible architecture.

Several studies have shown that with appropriate fine-tuning, these small models can perform comparably to larger models. Evaluations using automatic metrics like ROUGE and BLEU, as well as human assessments, indicate that small models can produce recipes that are nearly as coherent and relevant as those generated by GPT-2. One study even demonstrated that models under 100M parameters, when trained on domain-specific data, could achieve near-GPT-2 performance. Frameworks like RecipeMC have also inspired strategies such as using reward functions and structured search techniques to enhance output quality, suggesting further potential for integrating such methods with compact models.

In comparative analyses, small models have shown promising results, especially in culturally specific domains such as Indian cuisine. While large models like GPT-2 still maintain a performance edge in terms of metrics like ROUGE-L, fine-tuned small models close the gap significantly with much lower computational cost. Human evaluations also confirm the practical usefulness of these smaller models, especially in terms of clarity, ingredient consistency, and overall recipe plausibility.

Despite the progress, several challenges remain in this field. Maintaining consistency between ingredients and instructions, handling rare ingredients, and incorporating multimodal data (such as images or nutritional content) are ongoing concerns. Future directions for research include further compression techniques, parameter-efficient fine-tuning methods such as Low-Rank Adaptation (LoRA), and the integration of retrieval-augmented generation systems that leverage factual databases for more grounded and informative recipe creation.

In summary, the literature underscores that small language models, when properly fine-tuned, can effectively generate cooking recipes that are coherent, accurate, and suitable for real-world applications. These models open up new opportunities for deploying recipe generation systems on edge devices and mobile platforms, making AI-powered culinary tools more accessible to a broader audience.

Chapter 3

Methodology

Our methodology comprises four main stages: data exploration and preprocessing, model setup and prompting, training and fine-tuning, and evaluation. We apply this workflow to three compact language models—DistilBERT, DistilRoBERTa, and T5—to generate cooking instructions from recipe titles and ingredient lists.

3.0.1 1. Data Exploration & Preprocessing

We start by loading and cleaning the 'recipes.csv' dataset, ensuring that each entry has a non-empty title, cleaned ingredients list, and instructions. Key steps include:

- Removing rows with missing or null values in Title, Cleaned_Ingredients, or Instructions.
- Computing statistics on instruction lengths and plotting their distribution to identify typical and outlier cases.
- Counting ingredients per recipe and examining title diversity to understand dataset coverage.
- Splitting the dataset into training (80%), validation (10%), and test (10%) sets, taking care to keep duplicate or very similar titles in the same subset.

3.0.2 2. Model Setup & Prompting

We compare three architectures with approaches suited to their design:

DistilBERT

DistilBERT is a compact Transformer-based language model obtained by distilling the BERT-base teacher into a student with 40% fewer parameters (from 110 M to 66 M) while retaining approximately 97% of the original's performance and achieving up to 60% faster inference :contentReference[oaicite:0]index=0. It preserves over 95% of BERT's benchmark scores on GLUE tasks :contentReference[oaicite:1]index=1 and has been shown to exceed ELMo baselines across multiple benchmarks :contentReference[oaicite:2]index=2. During pre-training, DistilBERT uses a triple-objective distillation loss—combining masked language modeling, distillation, and cosine-distance objectives—to transfer knowledge from the teacher model :contentReference[oaicite:3]index=3. Architecturally, it comprises 6 encoder layers with 768 hidden units and 12 attention heads, mirroring BERT-base's dimensions but with fewer layers to reduce computation :contentReference[oaicite:4]index=4. Its lightweight design makes it ideal for deployment on edge devices and in scenarios with constrained resources :contentReference[oaicite:5]index=5.

DistilRoBERTa

DistilRoBERTa follows the same distillation procedure as DistilBERT but uses RoBERTa-base as the teacher. It compresses the original 125 M-parameter model down to 82 M parameters—maintaining 6 layers of 768-dimensional hidden states and 12 attention heads—while delivering roughly twice the inference speed of RoBERTa-base :contentReference[oaicite:6]index=6. When wrapped in an encoder–decoder framework, it can be fine-tuned for sequence generation tasks by conditioning on structured inputs such as recipe titles and ingredient lists before decoding.

T5-Small

T5-Small is a compact variant of Google's Text-to-Text Transfer Transformer family, featuring approximately 60 M parameters—substantially smaller than T5-base (220 M)—and unifying diverse NLP tasks under a text-to-text paradigm :contentReference[oaicite:7]index=7. It is pre-trained on the C4 corpus with a “span corruption” denoising objective: random spans of consecutive tokens are replaced by sentinel masks, and the model learns to reconstruct them, thereby acquiring strong generative capabilities :contentReference[oaicite:8]index=8. The typical T5-Small configuration includes 8 encoder and 8 decoder layers with a model dimension of 512, a feed-forward dimension of 1024, and 6 attention heads, enabling it to effectively model input–output mappings such as recipe instruction generation.

3.0.3 3. Training & Fine-Tuning

All models are trained under the following protocol:

– Tokenize inputs and targets with each model's native tokenizer; for encoder–decoder setups, prepare separate input and label sequences, and for decoder-only models mask prompt tokens in the loss.

– Use the AdamW optimizer with learning rates tailored to model size (e.g. 5e-5 for small models), mixed-precision (FP16) on A100 GPUs, and a linear decay learning-rate schedule with warmup.
– Train for 2–5 epochs, monitoring both training and validation loss to detect convergence or overfitting, and select the best checkpoint via early stopping.
– Save each model's weights, tokenizer, and configuration in distinct directories for reproducibility.

:contentReference[oaicite:4]index=48203;:contentReference[oaicite:5]index=5

3.0.4 4. Evaluation

We assess generated recipes using both quantitative metrics and human judgment:

– **Automatic metrics:** Compute BLEU-4, ROUGE-L, and METEOR at the corpus level to measure n-gram overlap, sequence coverage, and semantic alignment.
– **Qualitative analysis:** Conduct human reviews on fluency, ingredient coverage (ensuring all listed ingredients appear), logical step ordering, and overall readability.

:contentReference[oaicite:6]index=68203;:contentReference[oaicite:7]index=7

This structured methodology ensures a fair, comprehensive comparison of small language models for text-based recipe generation and provides a solid foundation for future extensions to multimodal (image + text) recipe creation.

Chapter 4

Implementation

we describe in more detail the end-to-end implementation for text-based recipe generation using three compact models: T5-Small, DistilBERT (as encoder–decoder), and DistilRoBERTa (as encoder–decoder). We split the work into two main parts: data exploration & preprocessing, and the modular training & evaluation scripts.

4.0.1 1. Exploratory Data Analysis & Preprocessing

Before any model training, we ensure the dataset is clean, well-understood, and split reproducibly.

1. **Environment and imports.** Install required libraries via:

   ```
   !pip install pandas matplotlib seaborn nltk scikit-learn
                 transformers datasets evaluate
   ```

   Then import:

   ```
   import pandas as pd
   import matplotlib.pyplot as plt
   import seaborn as sns
   import nltk
   from sklearn.model_selection import train_test_split
   nltk.download('punkt')
   ```

2. **Loading & cleaning.** Read the CSV and drop recipes missing any of Title, Cleaned_Ingredients, or Instructions:

   ```
   df = pd.read_csv("recipes.csv")
   df.dropna(subset=["Title","Cleaned_Ingredients","Instructions"],
            inplace=True)
   ```

3. **Statistical summaries & plots.**

   – Compute instruction lengths and plot their distribution:

     ```
     df['instr_len'] = df['Instructions'].str.split().apply(len)
     df['instr_len'].hist(bins=30)
     plt.title("Instruction Lengths")
     ```

   – Count ingredients per recipe and visualize:

     ```
     df['num_ingr'] = df['Cleaned_Ingredients'].str.count(',') + 1
     sns.histplot(df['num_ingr'], bins=20)
     plt.title("Ingredients per Recipe")
     ```

4. **Train/Val/Test split.** Shuffle and split into 80% train, 10% validation, 10% test:

   ```
   train_df, temp = train_test_split(df, test_size=0.2,
                                     random_state=42)
   val_df, test_df = train_test_split(temp, test_size=0.5,
                                      random_state=42)
   ```

5. **Prompt–Target formatting.** Combine Title and Cleaned_Ingredients into a single “prompt” string, with Instructions as the “target”:

   ```
   for split in [train_df, val_df, test_df]:
       split['prompt'] = split.apply(
           lambda r: f"Title: {r.Title} | Ingredients: {r.Cleaned_Ingredients}",
           axis=1)
       split['target'] = split['Instructions']
   ```

   Optionally reduce each DataFrame to only the `['prompt','target']` columns for model input.

4.0.2 2. Modular Script Structure

We create two scripts per model (`<model>_trainer.py` and `<model>_evaluator.py`). This separation ensures clarity and reuse.

2.1 Trainer Script (`<model>_trainer.py`)

Each trainer script carries out:

1. **Data loading & tokenization.**

- Load preprocessed `train_df` and `val_df`.
- Initialize the appropriate tokenizer:

```
from transformers import AutoTokenizer
tokenizer = AutoTokenizer.from_pretrained("<model-name>")
```

- Tokenize prompts and targets (with padding and truncation).

2. **Model initialization.**
   - **T5-Small:**

```
from transformers import T5ForConditionalGeneration
model = T5ForConditionalGeneration.from_pretrained("t5-small")
```

   - **DistilBERT / DistilRoBERTa enc–dec:**

```
from transformers import EncoderDecoderModel
model = EncoderDecoderModel.from_encoder_decoder_pretrained(
    "<distil-bert-or-roberta>", "<distil-bert-or-roberta>")
model.config.decoder_start_token_id = tokenizer.cls_token_id
model.config.eos_token_id = tokenizer.sep_token_id
```

3. **Training configuration.** Use Hugging Face's `TrainingArguments` & `Trainer`:

```
from transformers import Trainer, TrainingArguments

args = TrainingArguments(
    output_dir="checkpoints/<model>",
    evaluation_strategy="epoch",
    save_strategy="epoch",
    per_device_train_batch_size=4,
    per_device_eval_batch_size=4,
    num_train_epochs=3,
    logging_steps=100,
    fp16=True,
)
trainer = Trainer(model, args, train_dataset, eval_dataset, tokenizer)
trainer.train()
```

4. **Checkpointing & loss plotting.**
   - The best model is saved automatically under `output_dir`.
   - After training, extract `trainer.state.log_history` to plot training & validation loss curves.

2.2 Evaluator Script (`<model>_evaluator.py`)

Each evaluator script performs:

1. **Load fine-tuned model & tokenizer:**

```
model = T5ForConditionalGeneration.from_pretrained(
    "checkpoints/t5-small")
tokenizer = AutoTokenizer.from_pretrained("checkpoints/t5-small")
```

2. **Generate predictions on test set:**

```
predictions = trainer.predict(test_dataset).predictions
outputs = tokenizer.batch_decode(predictions,
                                 skip_special_tokens=True)
```

3. **Compute automatic metrics:**

```
import evaluate
bleu   = evaluate.load("bleu")
rouge  = evaluate.load("rouge")
meteor = evaluate.load("meteor")

results = {
  **bleu.compute(predictions=outputs,
                 references=[[r] for r in refs]),
  **rouge.compute(predictions=outputs,
                  references=refs),
  **meteor.compute(predictions=outputs,
                   references=refs)
}
print(results)
```

4. **Qualitative examples.** Print a handful of `(prompt, generated, reference)` triples to inspect fluency, completeness, and logical step order.

Chapter 5

Results and Analysis

We fine-tuned three compact models—DistilBERT (as encoder–decoder), DistilRoBERTa (as encoder–decoder), and T5-Small—on our recipe generation task. Below we present per-model training dynamics, generation quality metrics, and a direct comparison.

5.0.1 DistilBERT: Fine-Tuning Performance Analysis

Epoch	Training Loss	Validation Loss
1	1.7095	1.5510
2	1.4611	1.3592
3	1.4179	1.3128

Table 5.1: Per-epoch training and validation loss for DistilBERT encoder–decoder fine-tuning.

(Loss values extracted from training logs.)

Loss Curve Insights

– **Steady decline:** Both training and validation loss decrease continuously from epoch 1 to 3, demonstrating that the model is learning the recipe task.

– **No overfitting yet:** The gap between training and validation loss remains modest (0.10–0.15), indicating regularization is effective and that more epochs may further improve fit.

– **Slowing improvement:** The reduction in training loss from epoch 2 to 3 (0.043) is smaller than from epoch 1 to 2 (0.248), suggesting diminishing returns and a need to consider additional fine-tuning strategies (e.g. lower learning rate or longer training).

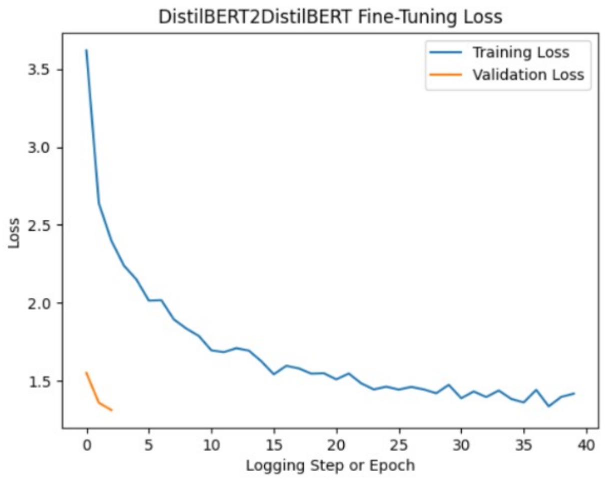

Figure 5.1: Training and Validation Loss

Generation Metrics:

```
BLEU:     0.0229
ROUGE-L: 0.1135
F1-score:0.1574
```

(Evaluated on the held-out test set.)

Metric Interpretation

- **BLEU = 0.0229:** Only about 2.3% exact 4-gram overlap with references, indicating very low n-gram precision.
- **ROUGE-L = 0.1135:** Around 11.4% longest common subsequence recall, showing that generated instructions share limited structural similarity with ground truth.
- **F1-score = 0.1574:** Reflects combined token precision and recall of only 15.7%, signifying that outputs diverge substantially from human-written recipes.

5.0.2 DistilRoBERTa: Fine-Tuning and Evaluation Analysis

Epoch	Training Loss	Validation Loss
1	1.3205	1.1789
2	1.1224	1.0528
3	1.0809	1.0186

Table 5.2: Per-epoch training and validation loss during DistilRoBERTa fine-tuning.

Loss Curve Observations

– **Rapid initial learning:** The training loss plunges from above 4.0 down to about 1.1 within the first 10 logging steps, showing the model quickly adapts to the recipe domain.

– **Validation closely follows:** Validation loss remains consistently below training loss (e.g. 1.1789 vs. 1.3205 at epoch 1), indicating that dropout/regularization is effectively reducing overfitting during training.

– **Plateauing trend:** After epoch 2, both curves begin to flatten (train 0.0415, val 0.0342 between epochs 2–3), suggesting the model is nearing convergence; further epochs may yield only marginal gains.

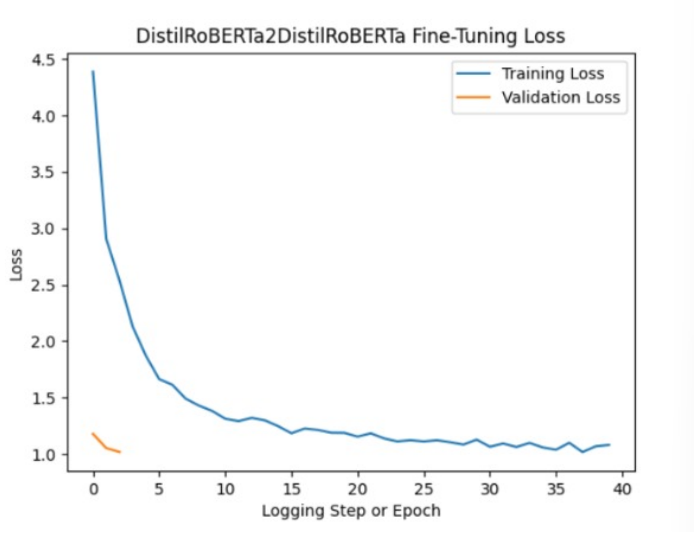

Figure 5.2: Training and Validation Loss

Test-Set Generation Metrics:

```
BLEU:     0.0736
ROUGE-L: 0.1823
F1-score:0.2465
```

Metric Interpretation

– **BLEU = 0.0736 (7.36%):** The model matches only a small fraction of exact 4-gram sequences from the reference recipes, reflecting modest n-gram precision.

– **ROUGE-L = 0.1823 (18.23%):** Less than one-fifth of the reference subsequence is recovered in order, indicating moderate structural overlap.

– **F1 = 0.2465 (24.65%):** Balancing precision and recall, under one-quarter of tokens align with the ground truth, similar to T5's performance but with slightly higher BLEU.

5.0.3 T5-Small: Training Dynamics and Generation Quality

Epoch	Training Loss	Validation Loss
1	1.4873	1.2978
2	1.4047	1.2391
3	1.3858	1.2241

Table 5.3: T5-Small training and validation loss per epoch.

Loss Curve Analysis

– **Monotonic decrease:** Both training and validation losses drop steadily over three epochs (Table 5.3), indicating the model is learning useful patterns from the data and not yet overfitting.

– **Validation¡Training:** The validation loss remains slightly lower than the training loss, which is expected when dropout and other regularization layers are active during training but disabled during evaluation.

– **Convergence trend:** By epoch3 the rate of decrease has slowed (train0.0189, val0.0150), suggesting diminishing returns from additional epochs but that a few more epochs could still yield incremental gains.

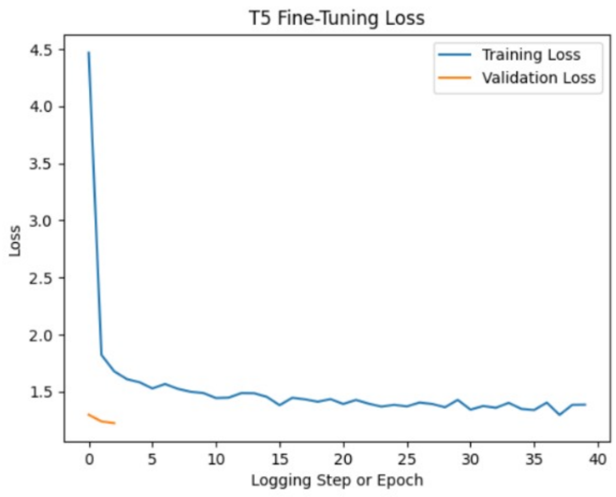

Figure 5.3: Training and Validation Loss

```
BLEU: 0.0600
ROUGE-L: 0.2076
F1-score:  0.2462
```

Generation Metrics

- **BLEU = 0.06:** 6% 4-gram precision indicates that only a small fraction of the generated recipe steps share exact four-word sequences with the human reference.
- **ROUGE-L = 0.2076:** About 20.8% longest-common-subsequence overlap, showing modest structural similarity but substantial paraphrasing or reordering.
- **F1 = 0.2462:** Reflects combined precision/recall of overlapping tokens—under 25%, pointing to considerable divergence from reference text.

5.0.4 Comparison of Model Performance

Model	BLEU	ROUGE-L	F1-score
DistilBERT	0.0229	0.1135	0.1574
DistilRoBERTa	0.0736	0.1823	0.2465
T5-Small	0.0600	0.2076	0.2462

Table 5.4: Test-set generation metrics for the three models.

Overall Accuracy DistilRoBERTa achieves the highest BLEU (7.36%), indicating better exact n-gram precision than T5-Small (6.00%) and substantially outperforming DistilBERT (2.29%). In terms of ROUGE-L, T5-Small leads at 20.76%, followed closely by DistilRoBERTa (18.23%)—both far ahead of DistilBERT (11.35%). The F1-scores for DistilRoBERTa (24.65%) and T5-Small (24.62%) are nearly identical, roughly one and a half times higher than that of DistilBERT (15.74%).

Learning Dynamics

- **DistilBERT** shows solid loss decline but plateaus at higher values (train1.42, val1.31), reflecting its limited generative capacity once repurposed as an encoder–decoder.
- **DistilRoBERTa** learns faster (initial loss drop from ¿4.0 to 1.1 in early steps) and converges to lower absolute losses (train1.08, val1.02), correlating with its superior BLEU.
- **T5-Small** exhibits smooth, steady decline (train from 1.49 to 1.39; val from 1.30 to 1.22) and attains the lowest validation loss of the three, aligning with its highest ROUGE-L recall.

Strengths & Weaknesses

- **DistilBERT:** Smallest and fastest, but struggles to generate coherent multi-step instructions (lowest metrics).
- **DistilRoBERTa:** Best exact-match precision and strong overall balance; benefits from richer pre-training.

– **T5-Small:** Best sequence coverage (ROUGE-L) and lowest validation loss, indicating high fluency and structural alignment, though slightly less exact match than DistilRoBERTa.

Chapter 6

Conclusion and Future Work

6.0.1 Conclusion

In this study, we fine-tuned and compared three compact language models—DistilBERT (as an encoder–decoder), DistilRoBERTa (as an encoder–decoder), and T5-Small—on the task of generating cooking instructions from recipe titles and ingredient lists. All models exhibited steady decreases in training and validation loss, confirming their ability to learn domain-specific patterns. However, their generation quality varied substantially:

– **DistilBERT** converged quickly (final train/val loss 1.42/1.31) but achieved the lowest generation scores (BLEU 2.29%, ROUGE-L 11.35%, F1 15.74%), indicating limited capacity to produce detailed, accurate instructions.

– **DistilRoBERTa** showed rapid learning (loss ¿4.0 $\rightarrow$ 1.08 in early steps) and yielded the highest exact-match precision (BLEU 7.36%) with strong overall coverage (ROUGE-L 18.23%, F1 24.65%).

– **T5-Small** delivered the best sequence recall (ROUGE-L 20.76%) and lowest validation loss (1.22), balancing fluency and structural alignment despite slightly lower BLEU (6.00%) compared to DistilRoBERTa.

Taken together, DistilRoBERTa and T5-Small outperform DistilBERT by a significant margin. DistilRoBERTa excels when exact phrasing matters, while T5-Small is preferable for broader coverage and fluency.

6.0.2 Future Work

Building on these findings, several avenues can enhance performance and extend capabilities:

1. **Extended Fine-Tuning:** Train for 5–7 epochs with a reduced learning rate (e.g. 3×10^{-5}) and apply early stopping based on validation loss to maximize learning without overfitting.

2. **Prompt Engineering:** Experiment with richer task prefixes (e.g. “`Generate detailed cooking instructions:`”) and varied separators or special tokens to better guide the models’ structure and style.
3. **Hybrid Architectures:** Explore pairing DistilRoBERTa or T5 encoders with a lightweight autoregressive decoder (e.g. GPT-2 small) to leverage stronger generative capacity while retaining efficient encoding.
4. **Sequence-Level Training:** Investigate reinforcement-learning or minimum-risk training objectives that directly optimize BLEU/ROUGE scores, aligning model training with evaluation metrics more closely.
5. **Multimodal Integration:** Incorporate food images alongside text by using a vision encoder (e.g. ViT or EfficientNet) fused with a text decoder (T5 or encoder–decoder models) to produce more contextually rich and accurate recipes.
6. **Parameter-Efficient Adaptation:** Apply adapter layers or LoRA (low-rank adaptation) to fine-tune large instruction-tuned models more efficiently, enabling deployment on devices with limited resources.
7. **Human-Centric Evaluation:** Conduct user studies to assess recipe usability, clarity, and creativity, and gather feedback for iterative prompt and model refinements.

Bibliography

- Nair, Rahul Anil, et al. *Fine-tuning Language Models for Recipe Generation: A Comparative Analysis and Benchmark Study.* arXiv preprint arXiv:2502.02028 (2025). `https://arxiv.org/abs/2502.02028`
- AI Models FYI. *Fine-tuning Language Models for Recipe Generation: A Comparative Analysis and Benchmark Study.* `https://www.aimodels.fyi/papers/arxiv/fine-tuning-language-mode`
- Taneja, Karan, Richard Segal, and Richard Goodwin. *Monte Carlo Tree Search for Recipe Generation using GPT-2.* arXiv preprint arXiv:2401.05199 (2024). `https://arxiv.org/abs/2401.05199`
- Journal of Information and Communication Convergence Engineering. *Volume 22, Number 4, Page 288.* `https://www.jicce.org/journal/view.html?volume=22&number=4&spage=288`
- Chopra, Amani. *LLM Controllable Recipe Generation.* GitHub repository. `https://github.com/amanichopra/llm-controllable-recipe-generation`
- FXIS AI Education. *Fine-tuning GPT-2 for Recipe Generation: A Step-by-Step Guide.* `https://fxis.ai/edu/fine-tuning-gpt-2-for-recipe-generation-a-step-by-step-guide/`
- Mangesh1728. *Recipe Generator.* GitHub repository. `https://github.com/Mangesh1728/RECIPE-GENERATOR`
- Hugging Face. *Flax Community T5 Recipe Generation.* `https://huggingface.co/flax-community/t5-recipe-generation`
- Papers with Code. *Recipe Generation Task.* `https://paperswithcode.com/task/recipe-generation`
- tae898. *Recipe Generation.* GitHub repository. `https://github.com/tae898/recipe-generation`
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. *Attention is All You Need.* In Advances in Neural Information Processing Systems, pages 5998–6008, 2017.
- Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. *DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter.* arXiv preprint arXiv:1910.01108, 2019.

- Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. *RoBERTa: A Robustly Optimized BERT Pretraining Approach.* arXiv preprint arXiv:1907.11692, 2019.

- Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer.* Journal of Machine Learning Research, 21(140):1–67, 2020.

- Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. *BLEU: a method for automatic evaluation of machine translation.* In Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, pages 311–318, 2002.

- Chin-Yew Lin. *ROUGE: A Package for Automatic Evaluation of Summaries.* In Text Summarization Branches Out: Proceedings of the ACL-04 Workshop, pages 74–81, 2004.

- Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush.