NextByte: Multi-Model Pipelines for Recipe Generation

Fynn Hayton-Ruffner
fhaytonruffner25@amherst.edu

Michael Xu
mjxu25@amherst.edu

Paul Vander Vort
pvandervort25@amherst.edu

Abstract

The most popular LLMs today are designed to be good at tasks ranging from computer programming to playwriting. Inspired by the idea that specialized expert models may outperform these general models, the prevalence of research into Mixture of Experts and fine-tuning smaller models has grown and yielded promising results. In this work, we compare the viability of two novel specialist model pipelines to a general-purpose model for recipe generation across a number of metrics. We discuss related work, provide an overview of the methods used, and detail an analysis of our results. We find that the mixture of a general purpose and specialist model improves recipe generation when compared to the general purpose model alone, while the specialist-only pipeline performs the worst overall.

1 Introduction

Since the introduction of ChatGPT in 2022, a growing number of companies are racing to produce high-quality Large Language Models (LLMs). LLM design today is driven by a key insight from the last few years: increasing scale improves performance. As a result, new LLMs have consistently been designed with a significantly larger dataset, vocabulary, and parameter count than their predecessors. Additionally, current LLMs are designed to be omniscient, with the best models as of May 2025 able to speak multiple languages, code in almost any programming language, and generate remarkably good responses to prompts spanning from college-level physics to creative writing (Minaee et al., 2024; Xia et al., 2025; Grattafiori et al., 2024).

While these results are undoubtedly impressive, the current paradigm is a far cry from how humans operate. Though humans often possess decent general knowledge about different fields, we tend to specialize in just a few areas. Additionally, we get by on just a couple of meals a day, significantly less than the $100+ billion data centers being built to fuel LLMs. This observation has inspired a lot of research into ways to achieve near state-of-the-art performance with more realistic time and energy requirements. One such approach is Mixture of Experts (MoE) in which only certain "experts" (segments of a model) are activated depending on the task. MoE LLMs have achieved impressive results in recent years at a fraction of the cost of larger LLMs like GPT-4 (Cai et al., 2024). Taking the underlying concept of MoE one step further, others have aimed to actually train a suite of small "specialist" models to conduct tasks that are inherently sequential or require multi-step reasoning (Fu et al., 2023).

Prior work training specialist models has been conducted on mathematical reasoning via the fine-tuning of relatively small pretrained language models (Fu et al., 2023). In this work, we examine the efficacy of specialist models trained from scratch for another inherently structured task: recipe generation. Rather than adopting an ensemble or MoE approach, we propose two pipeline architectures for integrating specialist contributions to a multi-step task: a strictly specialist design and one facilitated by a general knowledge model. To establish a baseline, one model was trained on each of the core components of a recipe: the title, ingredients, and directions. At the same time, we instantiated two more models: one tasked with producing ingredients from a title, the other responsible for generating directions from ingredients. During training, the performances of each individual model on their own tasks were assessed. In the post-training phase, the two specialists were constructed into the specialist-only pipeline, while the Title $\rightarrow$ Ingr model and the baseline formed the specialist-facilitated pipeline. The generation ability of each pipeline was evaluated against the baseline model both qualitatively and quantitatively.

While we expected the specialists to surpass the control on their training data, it was predicted that the "general knowledge" control would produce recipes that were more reflective of the data due to a loss-of-information effect within the specialist pipeline. Additionally, we believed the mixed pipeline would produce the highest quality recipes. Our results mostly followed these hypotheses. Specialists performed better than the generalist model at their specific tasks, the pipeline with only specialists performed quantitatively worse than the generalist model likely due to information loss, and the mixed pipeline performed better than either model on all metrics.

2 Related Work

2.1 Transformers

First proposed in 2017 in the now famous paper "Attention Is All You Need" (Vaswani et al., 2017), the Transformer has proven to be a very good architecture for NLP tasks. In the original paper, the Transformer contains both encoder and decoder structures. With this setup, the encoder takes some kind of data (like sentences) and outputs a contextualized embedding that is then fed into the decoder to produce a prediction (like the next token). However, for generative tasks like text completion, a decoder-only architecture is often used (Minaee et al., 2024). Given that we were concerned with recipe generation and did not want to limit prompts to a fixed title or a fixed set of ingredients, we chose to use a decoder-only architecture.

After embedding an input sequence, a positional encoding matrix PE is added to the input embedding to impart information about the position of each token and its corresponding values onto each vector. While there are many positional encoding formulas to choose from, we used:

$$PE_{(pos,2i)} = sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

$$PE_{(pos,2i+1)} = cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

Where pos is the position of the token in the input sequence, i is the dimension within the embedded vector, and d_{model} is the dimension of the model determined by the embedding layer.

The decoder structure in Transformers follows the general structure: repetition of multi-head attention and add and norm, feed-forward, linear, softmax. In multi-head attention, the input vectors are split into h different heads. Within each head, an input X of shape (seq_length, d_{model}), weight matrices W_{Q_i}, W_{K_i}, and W_{V_i} each with shape (d_{model}, d_{head}) are used to construct $Q_i = XW_{Q_i}$, $K_i = XW_{K_i}$, and $V_i = XW_{V_i}$. Here, seq_length is the length of the input sequence, and $d_{head} = \frac{d_{model}}{h}$. The output for each head is then calculated via:

$$\text{Attention}_i(Q_i, K_i, V_i) = \text{softmax}\left(\frac{Q_i K_i^T}{\sqrt{d_{head}}}\right) V_i$$

Finally, each Attention_i is concatenated to produce a matrix of shape (d_{model}, d_{model}) and multiplied by a weight matrix W_o also of shape (d_{model}, d_{model}). Formally:

$$\text{O}_{mha} = \text{concat}(\text{Attention}_o, \cdots, \text{Attention}_h)W_o$$

Where O_{mha} is the output of multi-head attention. In the add and norm step, this output is first added to the original input X and normalized using layer normalization which standardizes each vector in the input according to its mean and variance along the embedding dimension (Ba et al., 2016; Vaswani et al., 2017). For O_{an} = output from add and norm, we have:

$$\text{O}_{an} = \text{LayerNorm}(X + \text{O}_{mha})$$

In the feed-forward step, O_{an} is projected to a higher dimension using a string of standard linear layers before projecting the higher dimensional matrix back to (seq_length, d_{model}). Finally, a linear layer is applied to produce a matrix of shape $(seq\_length, vocab\_size)$ and softmax is applied to produce logits for each token in the vocabulary. For LLMs, multiple transformer decoders are often stacked on top of each other to enhance the model's ability to learn complex relationships (Vaswani et al., 2017).

2.2 LLMs and Specialists

LLMs have taken the world by storm in recent years. While there are many different implementations, the largest and most popular models like ChatGPT, Llama, Claude, and Gemini are all giant general-purpose chatbots trained on enormous amounts of data. Importantly, these models are not compartmentalized; rather, they aim to specialize in everything (Grattafiori et al., 2024; Minaee et al., 2024).

While not as popular, informed by the enormous inference and training costs of the largest LLMs, there have been a number of models designed to reduce computational requirements by switching between different "experts" or "specialists" depending on the task they are given. One such approach is called Mixture of Experts (MoE). MoE LLMs follow the same structure as Transformer decoders up until the feed-forward layers. Rather than feed the output from multi-head attention and add and norm directly into a single chain of linear layers, MoE models use a gating mechanism to send the output to the correct combination of feed-forward networks $\{f_1, \cdots, f_n\}$. These feed-forward networks are referred to as "experts". A weighted element-wise sum of the outputs from each expert is computed, and then an add and norm is performed between this weighted sum and the output from multi-head attention and add and norm. Both sparse and dense MoE networks have shown promise in recent years. Notably, Mixtral, DeepSeek-V3, and Lory have all achieved very good results using MoE approaches (Cai et al., 2024).

There are two broad methods of designing the gate in a MoE LLM: dense MoE and sparse MoE. In dense MoE, every expert network is activated and weighted according to the output from the learned gate function. For some output from multi-head attention and add and norm X, dense MoE will output:

$$O_{dmoe} = \sum_{i=1}^{n} g(x)_i f_i(x)$$

Where $g(x)_i$ = the i^{th} index of the vector output of $g(x)$ is a scalar value (usually the output of a softmax operation so that $s_i \in (0, 1)$), and f_i is an expert feed-forward network.

While dense MoE typically yields superior results to sparse MoE, it is also often just as computationally expensive as a standard Transformer-based LLM. Sparse MoE aims to address this by, for any task, only activating the top k experts as determined by the gating mechanism. This can be formalized via the following piecewise function:

$$\hat{g}(x)_i = \begin{cases} \hat{g}(x)_i & \text{if } \hat{g}(x)_i \text{ in top } k \text{ of } \hat{g}(x), \\ -\infty & \text{otherwise} \end{cases}$$

Where $\hat{g}$ is the output of the gating function prior to applying softmax. This way, once softmax is applied, any experts not in the top k will have a weight of softmax$(-\infty) \approx 0$ and will not be executed, significantly saving compute. Importantly, spare MoE networks can run into a load balancing problem where certain experts are used on a huge number of tasks and others are used very little, forcing some experts to learn too much and failing to leverage the learning capacity of other experts. To combat this, an auxiliary loss function is often added to MoE networks that punishes putting too much weight on certain experts (Cai et al., 2024).

Other approaches have taken the concept of dividing tasks even further by training entirely different models and using them as an ensemble during inference. One such approach aimed to improve the ability of smaller models to conduct chain-of-thought (CoT) reasoning for math. To achieve this, code-davinci-002, a fairly large OpenAI model, was used to generate 40 CoT solutions for tasks from GSM8K, a diverse math problem dataset, and those that generated correct solutions were used to fine-tune a suite of FlanT5 models with far fewer parameters than code-davinci-002.

After fine-tuning FlanT5 models on CoT for math problems, different models were tested on a number of datasets. On math-based datasets, all models experienced an increase of between 6.1 and 25.5 percent accuracy with one outlier performing 4.1 percent worse on SVAMP. However, on general datasets, all models saw a decrease of between 21.1 and 41.8 percent, indicating that, while it is possible to train smaller specialists for increased performance on certain tasks, it often comes at the cost of generalization (Fu et al., 2023).

2.3 Evaluating LLMs

Evaluating LLM performance is hard. Two completely different pieces of text can have almost identical meaning (e.g. "I walked my dog through the neighborhood" and "I took Scout around the block"), creativity is difficult to quantify and reward, and model temperature and sampling preferences can have a huge impact on generation quality. Because of this, a lot of work has been done to develop methods for evaluating LLMs. Five evaluation methods are relevant to this paper: BLEU, Precision, Recall, F1, and BERT score.

2.3.1 BLEU

Aimed to automate the process of human evaluation of language models, BLEU uses n-gram precision to evaluate how many n-grams from a piece of generated text match n-grams in the reference corpus.

Formally, a BLEU score is calculated as:

$$\text{BLEU} = \text{BP} \cdot e^{\left(\sum_{n=1}^{N} w_n \log p_n\right)}$$

Where

$$p_n = \frac{\sum_{c \in C} \sum_{n\text{-}gram \in c} Count_{clip}(n\text{-}gram)}{\sum_{c' \in C} \sum_{n\text{-}gram' \in c'} Count(n\text{-}gram')}$$

Here, each $w_n \in \mathbb{R}$, C is a set of candidate sentences, $Count_{clip}(n\text{-}gram)$ is the minimum between the number of times $n\text{-}gram$ appears in the candidate sentence and the number of times it appears in the reference sentences, and $Count(n\text{-}gram)$ is the number of times $n\text{-}gram$ appears in the candidate sentence. Also,

$$\text{BP} = \begin{cases} 1 & \text{if } c > r \\ e^{(\frac{1-r}{c})} & \text{if } c \leq r \end{cases}$$

Where c is the sum of the lengths (typically by tokens or words) of generated sentences and r is the sum of the lengths of reference sentences chosen to be the closest in length to their corresponding candidate sentence (Papineni et al., 2002).

It has been shown that using smoothing is often helpful for evaluations smaller than document-level. Relevant to this paper is method 2 from (Chen and Cherry, 2014) which simply adds 1 to each $n\text{-}gram$ count.

Prior work has shown BLEU to be an effective evaluation method, mapping closely to human evaluation and correlating well with perceived model performance. That said, BLEU does have drawbacks. Since BLEU counts n-grams, it does not account for synonyms and grammatical differences. These features mean that BLEU can end up punishing diverse text generation. Still, though, BLEU does give a good sense of how well generations match the training corpus and can be very useful, particularly when combined with other evaluation metrics (Papineni et al., 2002; Minaee et al., 2024).

2.3.2 Precision

Precision is a measure of how precise our positive predictions are. It is calculated as:

$$P = \frac{TP}{TP + FP}$$

Where TP is the number of true positives and FP is the number of false positives. Precision is a good measure of how precise a model is in its predictions, but it can fail in instances where, for example, the model only gives a positive prediction when it is almost absolutely certain. In a case like this, the model will be very precise, but it will also probably predict a lot of false negatives (Powers, 2015).

2.3.3 Recall

Recall addresses the aforementioned issues with Precision and is calculated by:

$$R = \frac{TP}{TP + FN}$$

Where TP is the number of true positives and FN is the number of false negatives. Put simply, Recall calculates the percent of positive cases that our model was able to predict correctly. Like Precision, Recall is not without issues. In a case where a model always predicts positive, it will have a recall of 1 but will also have a huge number of false positives (Powers, 2015).

2.3.4 F1

As described in the previous 2 sections, Precision and Recall complement each other's drawbacks well. Because of this, F1 score, the harmonic mean of Precision and Recall, is often favored over either of them individually. Formally:

$$F1 = \frac{2}{\frac{1}{P} + \frac{1}{R}} = \frac{2PR}{R + P}$$

Where P is Precision and R is Recall. While a more holistic measure, F1 score still suffers from some of the same kinds of problems as Precision and Recall. F1 struggles to account for imbalanced datasets, synonyms, and grammatical differences (Powers, 2015; scikit-learn developers).

2.4 BERT Score

BERT score is a modified version of precision, recall, and F1 that utilizes pretrained models to produce a more robust metric for correctness in a NLP context than exact token accuracy. To do this, candidate and reference text are passed through the model to generate a contextual embedding. For X = tokens from the generated text and Y = tokens from the true text, we calculate:

$$P_{BERT} = \frac{1}{|X|} \sum_{x_i \in X} \max_{y_i \in Y} x_i^T y_i$$

$$R_{BERT} = \frac{1}{|Y|} \sum_{y_i \in Y} \max_{x_i \in X} y_i^T x_i$$

Note that for any vectors x, y, $x^T y = |x||y|cos(\theta)$ where θ is the angle between x and y. Because vectors are pre-normalized, this measure is equivalent to "cosine similarity" = $\frac{x^T y}{|x||y|}$ which is a good measure of how semantically similar two tokens are for contextual embeddings. Thus, P_{BERT} is an average of the maximum similarity between each vector x_i in the generated text and all vectors y_i in the reference text. This essentially tells us how well our generated text aligns with tokens found in the reference text. Similarly, R_{BERT} is an average of the maximum similarity between each vector y_i in the reference text and all vectors x_i in the generated text. R_{BERT} measures how completely the generated text covers the tokens in the reference text. Like standard F1 score, $F1_{BERT}$ is the harmonic mean of P_{BERT} and R_{BERT}, i.e.

$$F1_{BERT} = \frac{2P_{BERT}R_{BERT}}{R_{BERT} + P_{BERT}}$$

(Zhang et al., 2019)

3 Methods

3.1 The Dataset

Each model made use of the RecipeNLG dataset, a free, public resource available here:

https://www.kaggle.com/datasets/paultimothymooney/recipenlg

As a compilation of nearly 2.3 million recipes sourced from numerous reputable cooking websites and standardized into clean CSV format, this dataset provides an ideal starting point for any recipe generation task. While containing extraneous metadata, we focused on the three columns in the data frame relevant to our work: title, ingredients, and directions. In this format, ingredients and directions are stored as Python lists of strings, making it a simple concatenation task to create a unified string representation of one recipe.

3.2 The NextByte Transformer Model

In line with the generative nature of the task at hand, the models tested here were constructed in accordance with the decoder architecture described in the related work section. Because our work was comparative by nature, we initialized all three models with consistent hyperparameters within a reasonable range for the task. Their values are available for reference in Table 1.

Hyperparameter	value
context_length	512
d_model	512
num_heads	8
num_decoders	2
num_hidden_layers	8
d_hidden	2048

Table 1: Model hyperparameters

3.3 Text Preprocessing

To streamline preprocessing, we created a custom dataset class inherited from the PyTorch Dataset module. The following procedure, which was applied upon each instantiation of this class, outlines the cleaning, reformatting, and tokenizing of the data specific to the structure that each model (Title $\rightarrow$ All, Title $\rightarrow$ Ingr, Ingr $\rightarrow$ Dir) expected. The cleaning step was consistent across all datasets, replacing "None" values with empty strings, retaining only alphanumeric and punctuation characters, and converting to lowercase via the use of regex.

After text normalization, the lists of ingredients and directions for each recipe were joined to create one string for each component. Because we were training specialists, it was necessary to format the recipes differently for each model: Title $\rightarrow$ All combined all steps in the process, appending a <end_title> token after the title text, a <end_ingredients> token after the ingredients, and a <end> token after the instructions. Naturally, Title $\rightarrow$ Ingr kept only the content of the title and ingredients columns, while the title was removed from the format of the Ingr $\rightarrow$ Dir dataset. These distinctions are visualized in Table 2. It should be noted that, for the Title $\rightarrow$ Ingr model, the <end> token was appended to the end of the ingredients in place of <end_ingredients>. This was done to ensure that the end of the ingredients was treated as the end of the recipe, thereby aligning the training data of that model with its unique task of producing only ingredients from a title. Finally, these formatted strings were concatenated to form one cohesive representation that would serve as one input example, once tokenized, to the models.

A separate tokenizer was then trained from scratch on each dataset's uniquely formatted recipe strings. We used a WordPiece sub-word tokenizer from the HuggingFace transformers library, with a vocabulary of 20,000 tokens. It was observed

Model	Recipe Format
Title -> All	...<end_title> ...<end_ingredients> ...<end>
Title -> Ingredients	...<end_title> ...<end>
Ingredients -> Instructions	...<end_ingredients> ...<end>

Table 2: Model-specific recipe string format prior to tokenization

that this vocabulary size was more than sufficient to cover all characters and word-level tokens in the training data. For each tokenizer, the standard [PAD] (padding) and [UNK] (unknown) special tokens were included in addition to our custom tokens (Table 2) used to delineate distinct components of a recipe. The resulting tokenizers were saved and passed to their respective datasets so that recipe strings could be tokenized on the fly at data loading time.

To further demonstrate its modular nature, the entire process described above was achieved through one constructor call, where the *mode* argument specified the text format to be produced for each model:

> TokenizedRecipeNLGDataset(df, tokenizer, mode='all')

With a unique tokenizer for each model, we set up the data for training. To do this, the original data file from Kaggle was loaded into a Python environment, split into 70% train, 15% validation, and 15% test segments, transformed into the specific custom tokenized dataset expected by each model, and batched together with the corresponding next token labels. This process was repeated for each model and its respective dataset.

3.4 Training & Evaluation

Similar to model instantiation, we set global training parameters to ensure consistency for comparisons. AdamW with a learning rate of $3e^{-5}$ was used to optimize the Cross-Entropy loss function. Recipes were grouped into batches of 16, and each model was trained for 8 epochs. At the end of each epoch, next-token prediction for the entirety of the training and validation sets was evaluated by accuracy, average loss, and F1 (scikit-learn developers). Following the completion of the training loop, these same metrics were used to assess performance on the test set, and the model states were saved for post-training generative evaluation. The training programs were executed in parallel on separate GPUs belonging to FrostByte, Amherst College's high-performance computing cluster. The comprehensive training process lasted approximately four days, making extensive hyperparameter tuning for increased performance unfeasible in the given time frame. Fortunately, due again to the comparative nature of our research, exceeding the state-of-the-art was not required.

In the post-training phase, we analyzed each model's generative ability. It is here that a sharp contrast must be drawn between which models were evaluated during training and which were evaluated during generation. To accurately assess the performance of our proposed pipeline architectures against the Title $\rightarrow$ All model, the Title $\rightarrow$ Ingr and Ingr $\rightarrow$ Dir models were aggregated into a specialist-only pipeline we called "Seq", while Ingr $\rightarrow$ Dir spearheaded a pipeline aptly called "Mix" that fed into Title $\rightarrow$ All. The generation of all three followed the standard auto-regressive procedure used for decoder transformers, though an extra step was necessary to feed the output of the first model in each pipeline to the second.

In line with much of previous recipe NLG research, the trained models' generative ability was evaluated as a function of BLEU score as well as precision, recall, and F1 BERT scores using BERT base uncased, an LLM trained on a large corpus of English text. To contrast All, Seq, and Mix, we prompted each with the same 10,000 recipe titles sampled from RecipeNLG and compared the resulting generations both quantitatively against the references and qualitatively against one another.

4 Results

4.1 Training Performance

For an exhaustive picture of the results of the entire training process, including each model's by-epoch performance on training and validation as well as their corresponding performance on the test set, refer to Appendix A. Within this section, we discuss only the validation results but note that no serious signs of overfitting were observed for any model when examining the metric scores on the training, validation, and test sets.

Each model performed fairly well in accuracy (Fig. 1). The learning curves of Title $\rightarrow$ All and Ingr $\rightarrow$ Dir mirrored one another, finishing at 84%

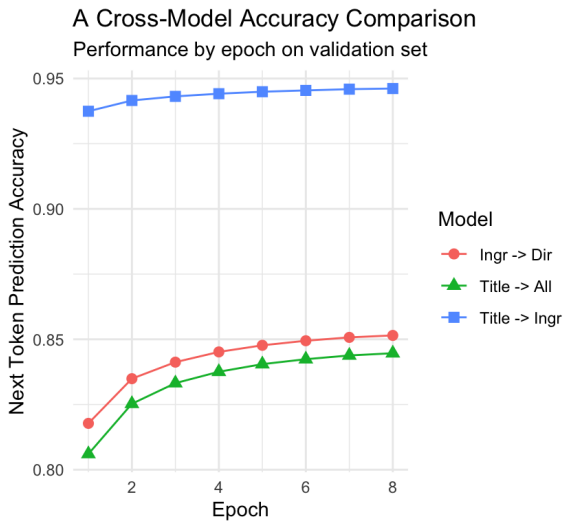

Figure 1: Validation accuracy by epoch for all models.

Figure 2: Validation average loss by epoch for all models.

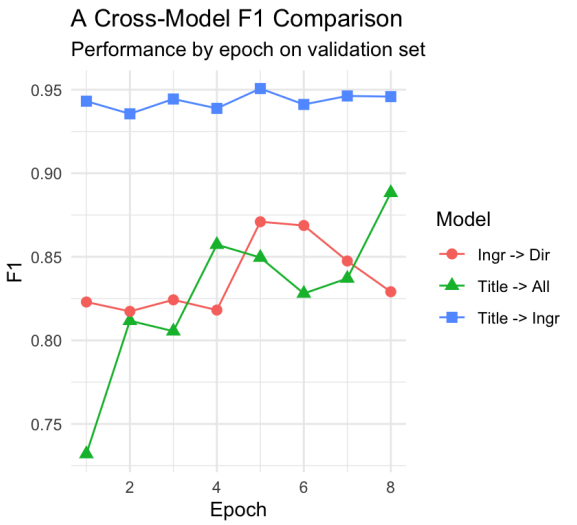

Figure 3: Validation f1 score by epoch for all models.

and 85% accuracy respectively. Title → Ingr strikingly outperformed its peers across all epochs, with a final validation accuracy of 94%.

The trends in average batch loss per epoch perfectly reflect the accuracy results. Ingr → Dir had slightly lower loss than Title → All after the final iteration, (0.68 versus 0.72), but the trend of their by-epoch loss reduction was identical (Fig. 2). Title → Ingr performed substantially better from the start (0.32), finishing at 0.25.

A slight deviation from the trends discussed was observed with the F1 evaluation. Although Title → Ingr performed excellently throughout to finish at 94%, the by-epoch F1 of the other two models was rather noisy and inconclusive. While both increased initially, they began to fluctuate somewhat after the fourth epoch (Fig. 3). Title → All (88%) ended higher than Ingr → Dir (82%), but the inconsistency of both models' per-epoch improvement makes any strong conclusion unlikely. Notably, Ingr → Dir performed much better on its test set (86% vs. 79% for Title → All).

4.2 All, Mix, & Seq: Recipe Generation

The average performance of each model's recipe generation on the 10,000 samples is shown in Table 3. There was no model overlap by any metric. Mix outperformed All and All outperformed Seq. To test the significance of these differences, we conducted a series of hypothesis tests, beginning with four analyses of variance (ANOVA) tests to identify the existence of at least one model-model difference across the metrics.

A one-way ANOVA revealed that there was a

Model	Samples	BLEU (avg)	Precision (avg)	Recall (avg)	F1 (avg)
All	10,000	0.059	0.637	0.621	0.627
Mix	10,000	0.060	0.643	0.623	0.631
Seq	10,000	0.058	0.634	0.619	0.624
		p < 0.001***	p < 0.001***	p < 0.001***	p < 0.001***

Table 3: Summary statistics with ANOVA results comparing trained model recipe generation

significant difference in BLEU score between at least two of the models (F(2, 29,997) = 11.79, p < 0.001). Similarly, an ANOVA fitted to precision uncovered a significant difference between at least two models (F(2, 29,997) = 57.18, p < 0.001). The same effect of model type was found for recall (F(2, 29,997) = 12.58, p < 0.001) and F1 (F(2, 29,997) = 42.42, p < 0.001). Clearly, at least two models performed differently on each metric. As a result, it was necessary to conduct post-hoc tests to look at pairwise model differences.

Four Tukey's Honest Significant Difference (HSD) tests were performed to examine pairwise model differences across each metric, the full results of which are available in Appendix B. The first test found that Mix (M = 0.060, SD = 0.038) significantly outperformed Seq (M = 0.058, SD = 0.038) on BLEU (p < 0.001, 95% CI = [-0.004, -0.001]), as did All (M = 0.059, SD = 0.039) (p < 0.01, 95% CI = [-0.003, -0.0005]). On precision, Mix (M = 0.643, SD = 0.059) performed significantly better than All (M = 0.637, SD = 0.060) (p < 0.001, 95% CI = [0.004, 0.008]) and Seq (M = 0.633, SD = 0.059) (p < 0.001, 95% CI = [-0.011, -0.007]). All also exceeded the scores of Seq (p < 0.001, 95% CI = [-0.005, -0.001]). Similar to BLEU score, Mix (M = 0.622, SD = 0.058) was better than Seq (M = 0.618, SD = 0.059) (p < 0.001, 95% CI = [-0.006, -0.002]) on recall, as was All (M = 0.621, SD = 0.059) (p < 0.01, 95% CI = [-0.004, -0.001]). Finally, Mix (M = 0.631, SD = 0.049) generated higher quality recipes by F1 than All (M = 0.627, SD = 0.051) (p < 0.001, 95% CI = [0.001, 0.005]) and Seq (M = 0.624, SD = 0.049) (p < 0.001, 95% CI = [-0.008, -0.004]). As was the trend on other metrics, All did better than Seq as well (p < 0.001, 95% CI = [-0.004, -0.001]).

Definitively, Mix and All were superior models for the top-to-bottom recipe generation task. Mix outperformed All on every metric and significantly so for precision and recall.

5 Discussion

Individual model performance is in line with prior work in this space. Like in (Fu et al., 2023), specialist models outperform generalists. In our case, Title → Ingr performed significantly better than Title → All at generating titles and ingredients. Similarly, Ingr → Dir performed slightly better than Title → All at generating ingredients and directions. However, quantitative analysis of the specialist model pipeline indicated an inability to leverage the performance advantages that the individual models provided. That said, qualitative analysis did seem to indicate superior performance on individual tasks. Given the same prompt, ingredients and directions generated by Seq are often more coherent, detailed, and creative than those generated by All. Importantly, though, recipes from Seq do not always follow well from the title.

Given that Ingr → Dir was trained without a title, it cannot be seeded with a title and thus has no information about the recipe name. Still, though, Seq is not always wrong. Very often it will generate directions for the correct recipe despite Ingr → Dir having no information about the title. Furthermore, despite performing worse on the task of going from a title to an entire recipe, given that Ingr → Dir performed better than Title → All on its specific task, it is not entirely useless. If a user's prompt contained only ingredients (for example, some of the foods lying around in their fridge) without a specific dish in mind, Ingr → Dir would be a better model to use.

This work both reaffirms the idea that training models for specific tasks yields superior results to generalist models as well as dissuades experiments with multi-model pipelines constructed purely from specialists. The relatively poor performance of Seq is evidence of a "communication" issue between the two component models; with no notion of how a title contextualizes instructions, Ingr → Dir tends to lose the plot when it creates recipes. This result mirrors the natural flow of creation in the real world. In a multi-stage project, you often see spe-

cialized individuals assigned to their own parts or tasks. While those tasks tend to be completed with the quality only such skilled individuals could provide, without oversight, the result as a whole is often left incoherent, confusing, and aimless. The manner by which Mix, a pipeline aggregation of a specialist and general-purpose model, produced comparatively the best recipes by all metrics is an exciting result from this perspective. Title → All can be seen as the facilitator, guiding the output of the specialist it works with to produce a superior result. All this to say that the validity of the proposed pipeline architecture is particularly legitimate in the present work not when created linearly, but hierarchically.

With this in mind, there are a number of interesting directions for future work exploring a hierarchical pipeline and to what extent increasing the number of small specialist models improves outcomes. Prior work has shown that CoT reasoning for math can occur in smaller, fine-tuned models (Fu et al., 2023). It would be interesting to see if small, highly specialized CoT models incorporated into an LLM pipeline would improve generations or at least yield similar results with significantly reduced compute. MoE approaches do something like an automatic search for a combination of specialist models, but each feed-forward network is trained in unison with the same tokens. It would be interesting to see future work looking at ways to automate the from-scratch tokenization and training of specialist models.

5.1 Limitations

The rationale for choosing recipe generation as the task to be evaluated is a potential hindrance to the generalization of these findings. A pipeline architecture was formulated because we were interested in sequential tasks, for which recipes are prototypical. By no means are all natural language generation tasks so straightforward, separable, and simple. Look no further than the training data. The process of going from title to ingredients to directions is built in such a way that it allows for objective separations between specialist tasks. In our tokenization process, it was easy to see where we would add the custom separator tokens. This will not always be the case. Thus, it is reasonable to question the extent to which our Mix pipeline may replicate its performance on assignments with more variety in their construction and process.

Though not a concern exclusive to this work, the viability of our quantitative analysis of the generated recipes cannot go unmentioned. In particular, BLEU has flaws as considerable as its prominent utilization in NLG evaluation projects. Using the geometric mean of the uni-, bi-, tri-, and quadgram precision generated text against a reference provides a shallow representation of the quality of said text. While it can certainly identify inchorence, it cannot by definition take into account synonyms, creativity, or detail. For example, consider the difference between "minced garlic" and "chopped garlic". If the model produced the minced version, why should it be penalized? Making "chopped" the required form of garlic is extremely restrictive. Synonyms like "scallion" and "green onion" would be similarly misunderstood.

Finally, the general time crunch of the current work should not be ignored. Most relevant to this consideration are the training and generation-analysis phases. With more time, it would have been possible to both train each model for longer than 8 epochs as well as conduct more exhaustive evaluations. BLEU comes to mind in this regard as well. The NLTK BLEU score allows for using multiple references when assessing generation. With more time, we could have identified recipes in the dataset that all make the same dish, collected those into a list of references, and compared model generation prompted with a generic title representation of that group. It is likely that, with a slew of possible ground-truth references, the BLEU metric would have been more lenient on the creative licenses taken by our specialist models. This method obviously would require combing through RecipeNLG extensively to find overlapping recipes, a task that was simply not feasible here.

5.2 Ethical Considerations

All too often, machine learning research pays little heed to the moral and social implications of its results. This issue is particularly true for natural language generation; as calls for ChatGPT to cite its sources grow, so too has unease around the over-reliance on such tools. Aside from fears of plagiarism and critical-thinking degradation, NLG projects can inspire cultural, financial, and safety concerns. It is within this context that we must situate the current work so as to push back on the trend of unaccountable AI.

In rare cases, chatbots designed to comfort a user as a convenient alternative to conventional therapy have generated exceptionally harmful and

damaging messages. Although concerns about psychological well-being are not paramount in current work, the same cannot be said for food safety. With a publicly accessible recipe generation model, it is crucial that it 1) can reliably filter its creations based on a user's allergenic profile and 2) always request such information before generation. We would not consider it outlandish to vet proposed cooking bots with an approval process similar to that of novel drugs, so inherent to this field is the necessity of user well-being. Even then, it would be advisable to provide a disclaimer clearly detailing the implications and risks of using an approved recipe NLG model. In our work, we are not proposing a model to be readily released to the public (at least not yet), but it is our hope that researchers in this field continue to take this issue seriously.

We must also acknowledge the cultural concerns for this work. RecipeNLG is a fantastic and relatively underused resource for recipe generation, but its quality does not necessarily justify the methods of its construction. The recipes it holds were scraped and processed from a wide range of cooking websites. While it is not for us to say if the robots.txt file of every single one of these websites consented to such extensive scraping, it is reasonable to assume that the millions of individuals who uploaded their creations to the web are unaware that their recipes are stored within a massive database, one that anyone with resources can take advantage of. This is where our work comes into play. Because they are trained on huge amounts of data, decoders tend to spit out a coherent yet diluted, mixed-up, and unrepresentative conglomerate of the primary sources that inform generation. In our example, when a cooking bot generates "Chicken Tikka Masala", it is likely that it is drawing on many such recipes in its training data, meaning the result will be a distorted combination of these recipes. Many chefs share familial, ancestral, or cultural recipes online that are of significance to their identity, but a cooking bot cannot, by definition, respect that importance. Misrepresentation of culture is thus an unavoidable evil in the current paradigm of NLG.

We do not claim to offer perfect solutions to the issues highlighted above, but it is our hope that continued discussion will fuel targeted research addressing resource accountability, potential physical and psychological harm from NLG models, and a higher awareness of the socio-cultural implications of their generation.

6 Conclusion

Our work supports future research examining the mixture of specialist and generalist models in a multi-model pipeline. We show that, for recipe generation, a pipeline consisting of general and specialist models is able to outperform the general model alone. By reducing the size of the Title $\rightarrow$ Ingr model, the Mix pipeline could presumably achieve comparable results to the general model while saving time and compute for close to half of the generated task. Previous work has shown that small, fine-tuned models can rival general purpose chat bots on the specific tasks that they were fine-tuned for (Minaee et al., 2024; Fu et al., 2023). By incorporating smaller specialist models and general-purpose chat bots into a single pipeline, it may be possible to achieve similar results to the current state-of-the-art with significantly reduced computation costs. Of course, the question of how to choose which tasks should be fine-tuned and how to jump between different models remains, giving rise to questions of whether a fully or semi-automated hierarchical approach would be best. Additionally, given that this work is limited to recipe generation and metrics like BLEU and BERT score are limited in their evaluation of generated text, future work would be necessary to test whether these results can generalize to other, less specialized tasks.

References

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer normalization.

Weilin Cai, Juyong Jiang, Fan Wang, Jing Tang, Sunghun Kim, and Jiayi Huang. 2024. A survey on mixture of experts in large language models.

Boxing Chen and Colin Cherry. 2014. A systematic comparison of smoothing techniques for sentence-level BLEU. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pages 362–367, Baltimore, Maryland, USA. Association for Computational Linguistics.

Yao Fu, Hao Peng, Litu Ou, Ashish Sabharwal, and Tushar Khot. 2023. Specializing smaller language models towards multi-step reasoning.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, and 1 others. 2024. The llama 3 herd of models.

Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. 2024. Large language models: A survey.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. 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, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

David M. W. Powers. 2015. What the f-measure doesn't measure: Features, flaws, fallacies and fixes.

scikit-learn developers. sklearn.metrics.f1_score. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010, Red Hook, NY, USA. Curran Associates Inc.

Zhiqiu Xia, Lang Zhu, Bingzhe Li, Feng Chen, Qiannan Li, Chunhua Liao, Feiyi Wang, and Hang Liu. 2025. Analyzing 16,193 llm papers for fun and profits.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. 2019. Bertscore: Evaluating text generation with bert.

A Training Results

	Accuracy		Avg Loss		F1 Score	
Epoch	Train	Val	Train	Val	Train	Val
1	0.8061	0.8060	1.4132	1.0266	0.7710	0.7320
2	0.8256	0.8253	0.9322	0.8687	0.8152	0.8117
3	0.8338	0.8332	0.8320	0.8057	0.8488	0.8055
4	0.8384	0.8376	0.7844	0.7715	0.8317	0.8573
5	0.8417	0.8405	0.7556	0.7496	0.8583	0.8496
6	0.8438	0.8424	0.7363	0.7357	0.8065	0.8280
7	0.8456	0.8438	0.7226	0.7263	0.8479	0.8371
8	0.8467	0.8447	0.7130	0.7213	0.8868	0.8883

(a) Title → All; Test Performance: accuracy = 0.8445, loss = 0.7210, F1 = 0.7941

	Accuracy		Avg Loss		F1 Score	
Epoch	Train	Val	Train	Val	Train	Val
1	0.9375	0.9374	0.4392	0.3208	0.9497	0.9431
2	0.9417	0.9416	0.2976	0.2833	0.9441	0.9356
3	0.9434	0.9431	0.2744	0.2687	0.9295	0.9444
4	0.9445	0.9442	0.2627	0.2603	0.9446	0.9388
5	0.9454	0.9449	0.2550	0.2543	0.9595	0.9507
6	0.9460	0.9454	0.2492	0.2505	0.9522	0.9412
7	0.9466	0.9459	0.2450	0.2477	0.9377	0.9462
8	0.9469	0.9462	0.2418	0.2463	0.9480	0.9459

(b) Title → Ingr; Test Performance: accuracy = 0.9461, loss = 0.2462, F1 = 0.9393

	Accuracy		Avg Loss		F1 Score	
Epoch	Train	Val	Train	Val	Train	Val
1	0.8179	0.8178	1.3142	0.9484	0.8152	0.8230
2	0.8353	0.8349	0.8619	0.8085	0.7520	0.8174
3	0.8420	0.8412	0.7775	0.7559	0.8896	0.8243
4	0.8462	0.8452	0.7359	0.7250	0.8527	0.8181
5	0.8490	0.8477	0.7098	0.7054	0.9072	0.8710
6	0.8511	0.8495	0.6919	0.6926	0.8831	0.8688
7	0.8527	0.8507	0.6791	0.6843	0.9080	0.8475
8	0.8537	0.8515	0.6701	0.6799	0.8971	0.8291

(c) Ingr → Dir; Test Performance: accuracy = 0.8515, loss = 0.6796, F1 = 0.8629

Figure 4: Model Performance During Training on Train, Val and Test

B Tukey's HSD Results for Model Comparisons

Contrast	Diff est.	Low	High	p adj.
mix-all	0.0008	-0.0005	0.0020	0.3381
seq-all	-0.0018	-0.0031	-0.0005	0.0026
seq-mix	-0.0026	-0.0038	-0.0013	0.0000

(a) Pairwise Model Differences on BLEU

Contrast	Diff est.	Low	High	p adj.
mix-all	0.0058	0.0038	0.0078	0e+00
seq-all	-0.0032	-0.0051	-0.0012	6e-04
seq-mix	-0.0089	-0.0109	-0.0069	0e+00

(b) Pairwise Model Differences on Precision

Contrast	Diff est.	Low	High	p adj.
mix-all	0.0012	-0.0007	0.0032	0.3040
seq-all	-0.0029	-0.0048	-0.0009	0.0018
seq-mix	-0.0041	-0.0061	-0.0021	0.0000

(c) Pairwise Model Differences on Recall

Contrast	Diff est.	Low	High	p adj.
mix-all	0.0035	0.0018	0.0051	0e+00
seq-all	-0.0031	-0.0047	-0.0014	1e-04
seq-mix	-0.0065	-0.0082	-0.0049	0e+00

(d) Pairwise Model Differences on F1

Figure 5: Results of Tukey's HSD for BLEU, Precision, Recall, and F1