INTRODUCTION:

Due to the abundance of cooking recipes available online, there is growing interest in harnessing this information to develop new recipes. In order to solve the "Novel Recipe Generation" problem, our primary goal is to create new, realistic cooking recipes that are so similar to the real ones that even culinary experts can't tell the difference. We fed a vast amount of structured recipe data into a deep learning model like GPT-2 in order to create these innovative recipes.

BACKGROUND STUDY:

We used the GPT2 model to generate novel recipes. GPT2 is essentially a machine learning language model that can anticipate the following word from a sentence's context (single token).
As part of their investigation, the Open 11 researchers searched the internet to gather a sizable 40GB dataset known as Web Text, which was used to train the GPT-2. The trained GPT-2's smallest variant requires 500 MB of storage to keep all of its parameters. Given that the largest GPT-2 model is 13 times larger, it may use more than 6.5 GB of storage. Transformer decoding blocks are used to construct GPT2. GPT2 outputs one token at a time. GPT2 is auto-regressive in nature; each token in the sentence has the context of the previous words. After each token is produced, that token is added to the sequence of inputs. And that new sequence becomes the input to the model in its next step. This is an idea called "auto-regression". The GPT-2 model can take into account sampling words other than the top word by using a parameter called top-k, which is the case when top-k = 1. While the GPT 3 model is trained on 175 billion parameters, the GPT2 model uses about 1.5 billion parameters.

Overview of feeding in text and generating a single token in GPT2:

1. Tokenization - Take some words, break them up into their common pieces. Take those common pieces and replace them with a number. Tokenization is necessary because computers only work with numbers. This also represents words efficiently.
2. Embedding with time signal - Take that 1 string of numbers and convert each number to a vector. This captures the position of words relative to one another and allows words to take value from other words associated with them, e.g. "The boy ran through the woods, and he surely had not stolen the cherry pie for which they were chasing him." In this sentence "he" should clearly tie a lot of its meaning to "boy".
3. Decoder Block - The pieces are self-attention blocks, feedforward neural nets, and Normalization. Self-attention blocks identify which words to focus on. In the sentence, "Jimmy played with the burning bush, and then went around to the next bush," the words "Jimmy," "played," "burning," and "bush" capture a high proportion of the meaning in that sentence. This idea that certain words and phases capture more meaning and thus should be given more "attention" is the intuition of self-attention blocks.
4. Linear Layer - Prior to the tokenization process, a vocabulary size will be decided upon and a vocabulary will be set up

5. . The vocabulary is just a list of all the possible tokens (numbers) that can be produced and which letter or group of letters the tokens are equal to. The linear layer takes the output of the last decoder block and converts it to a vector whose dimensions are vocabulary size by 1. In short, it takes a lot of inputs and produces a list where each spot represents a token. The higher the number in the spot the better the chance that that token is the best pick.
6. Softmax - Converts the output of the linear layer to a probability distribution. The output of the linear layer tells you information about which tokens are the best picks, but it is hard to use. The values range from very small values(huge negative values) to very large values and their meaning is in relation to all the other values. To make them easier to use apply the Softmax function which converts the vector to a probability distribution. This means each number represents the probability that that token is the correct one.
7. Pick a token - Choose the method to pick the next token from the probability distribution of tokens, and use that method to pick the token. There are various methods to do so including, greedy, temperature sampling, nucleus sampling, and top-k sampling. Convert Token to a word piece using the vocabulary.

METHODOLOGY:

DATASET:

We used the RecipeDB_v1 dataset which is an organized collection of recipes, ingredients, and nutrition profiles related to flavor profiles and health associations. The repertoire consists of the laborious fusion of over 1,18,000 recipes from cuisines around the world (six continents, 26 geo-cultural regions, and 74 countries), cooked with over 23,500 ingredients from various categories using 268 different processes (heat, cook, boil, simmer, bake, etc.), and further linked to their flavor molecules (FlavorDB), nutritional profiles (USDA), and empirical records of disease associations obtained from Medline (DietRx).
We used the file “Recipe_correct_ndb_updated_v1.csv” in the RecipeDB dataset to go ahead with our problem. The RecipeDB dataset consists of 1154404 records. There are 12 fields in the dataset, namely ‘recipe_no’, ‘ingredient_Phrase’, ‘ingredient’, ‘state’, ‘quantity’, ‘unit’, ‘temp’, ‘df’, ‘size’, ‘ing_id’, ‘ndb_id’, ‘M_or_A’. Out of all the fields, recipe_no, ingredient fields are used further to calculate the popularity of the ingredients across the recipes.

METHODS / WORKFLOW:

Data Preprocessing:

Following steps were performed for the data preprocessing:

1. Removal of recipes: Recipes which contain only a single ingredient are removed from the recipe database because they are not of much importance.

2. Merging of useful Information corresponding to recipes: Recipes are mapped with their corresponding instructions, region, continent, sub-region.

3. Lemmatization: Performed lemmatization on the ingredients set of all the recipes to convert all forms of each ingredient to their lemmatized form. Before performing this, all ingredient words are converted to lowercase so that the model should be able to analyze the same ingredients present in multiple recipes because we want that “potato” in one recipe and “Potato” in another recipe should be considered the same by the model.

4. Fixing Punctuations: This operation is also performed on the ingredients set of all recipes.

5. Removing Useless Symbols: Some symbols are present in recipe instructions that are not of any use, therefore we have removed all of these symbols from the instructions of all the recipes.

Tokenizer:

Along with the default tokens used by the GPT2 model, we have used the following tokens to differentiate different parts of the recipe data.

Token	**Denotes**
<BEGIN_RECIPE>	Start of the recipe
<BEGIN_INPUT>	Start of the input ingredient given
<NEXT_INPUT>	Next ingredient in the input ingredients
<END_INPUT>	End of the input ingredients
<BEGIN_TITLE>	Start of the recipe title
<END_TITLE>	End of the recipe title
<BEGIN_INGREDS>	Start of the recipe ingredients
<NEXT_INGREDS>	Next ingredient in the ingredient list
<END_INGREDS>	End of the recipe ingredients
<BEGIN_INSTR>	Start of the recipe instructions
<NEXT_INSTR>	Next instructions in the instructions
<END_INSTR>	End of the recipe instructions
<END_RECIPE>	End of the recipe

Model Training:

We have used the 118083 recipes data. Initially, we splitted the data into train and test sets where the train set contains 94.6 % and the test set contains 5.4% of the data. Then we attach the tokens defined above to each of the sets, as a result, train.txt and test.text files are generated. Then the model is trained using the recipes of the train data.
Following values of functional parameters of GPT 2 model are used to train the model:

```
num_train_epochs=2,
per_device_train_batch_size=4,
per_device_eval_batch_size=4,
gradient_accumulation_steps=8,
evaluation_strategy="steps",
fp16=True,
fp16_opt_level='O1',
warmup_steps=1e2,
learning_rate=5e-4,
adam_epsilon=1e-8,
weight_decay=0.01,
save_total_limit=1,
load_best_model_at_end=True
```

Model training had taken about 1 to 1.5 hours on system with server configuration mentioned in the results section and also it depends on the server load at the time of training:

Generation of recipes:

We used the RecipeDB data to train the GPT2 model, which we then used to produce new recipes. The list of ingredients was submitted to the GPT2 model. The method used to create the list of ingredients is listed below.

1. A random number from 2 and 8 was generated, and that many ingredients were chosen at random in proportion to how frequently they appeared across the recipes. The list of those ingredients was fed into the GPT2 model.

RANDOM INGREDIENTS:

kalamata olive,onion soup mix,romaine lettuce leaf,chocolate,beef bouillon granule,shiitake mushroom,yogurt

RECIPE TITLE:

Korean Beef and Shiitake Lettuce Wraps

INGREDIENT PHRASES:

1 cup plain yogurt | 1/4 cup grated miniature semisweet chocolate (such as kittencal's) | 3 tablespoons dried onions soup mix | 1 1/2 teaspoons beef bouillon granules | 8 -10 kalamata olives, rinsed and drained | 2 packages sliced shiitake mushrooms, stems removed (about 2 cups) | 1 small head romaine lettuce leaf

RECIPE INSTRUCTIONS:

In a bowl, stir together the yogurt, chocolate, onion soup mix, and beef bouillon granules. Set aside. To make slaw, in a bowl, stir together the olives, mushrooms, lettuce, and 1/3 the reserved yogurt mixture until evenly mixed. Top each wrap with the remaining yogurt mixture, spreading evenly. Serve immediately.

RESULTS:

We have generated **120k** recipes. The total time consumed to generate these recipes is **1200 hrs** when we generated recipes using the three instances of the same server.

Following is the server configuration:

```
# Name                Version          Build  Channel
_libgcc_mutex         0.1              conda_forge    conda-forge
_openmp_mutex         4.5              2_kmp_llvm    conda-forge
brotlipy              0.7.0            py38h27cfd23_1003
ca-certificates       2022.10.11       h06a4308_0
certifi               2022.9.24        py38h06a4308_0
cffi                  1.15.1           py38h74dc2b5_0
charset-normalizer    2.0.4            pyhd3eb1b0_0
click                 8.0.4            py38h06a4308_0
cryptography          38.0.1           py38h9ce1e76_0
cudatoolkit           10.2.89          h713d32c_10    conda-forge
cudnn                 7.6.5.32         h01f27c4_1    conda-forge
filelock              3.6.0            pyhd3eb1b0_0
future                0.18.2           pyhd8ed1ab_6    conda-forge
h5py                  3.7.0            py38h737f45e_0
hdf5                  1.10.6           h3ffc7dd_1
huggingface_hub       0.10.1           py38h06a4308_0
idna                  3.4              py38h06a4308_0
joblib                1.1.1            py38h06a4308_0
ld_impl_linux-64      2.38             h1181459_1
libblas               3.9.0            8_mkl    conda-forge
libcblas              3.9.0            8_mkl    conda-forge
```

```
libffi                  3.3             he6710b0_2
libgcc-ng               12.2.0          h65d4601_19    conda-forge
libgfortran-ng          11.2.0          h00389a5_1
libgfortran5            11.2.0          h1234567_1
liblapack               3.9.0           8_mkl    conda-forge
libstdcxx-ng            11.2.0          h1234567_1
llvm-openmp             14.0.6          h9e868ea_0
magma                   2.5.4           hfead8bd_4    conda-forge
mkl                     2020.4          h726a3e6_304    conda-forge
nccl                    2.14.3.1        h1a5f58c_0    conda-forge
ncurses                 6.3             h5eee18b_3
ninja                   1.11.0          h924138e_0    conda-forge
numpy                   1.22.3          py38h99721a1_2    conda-forge
openssl                 1.1.1q          h7f8727e_0
packaging               21.3            pyhd3eb1b0_0
pip                     22.2.2          py38h06a4308_0
pycparser               2.21            pyhd8ed1ab_0    conda-forge
pyopenssl               22.0.0          pyhd3eb1b0_0
pyparsing               3.0.9           py38h06a4308_0
pysocks                 1.7.1           py38h06a4308_0
python                  3.8.13          haa1d7c7_1
python_abi              3.8             2_cp38    conda-forge
pytorch                 1.7.1           cuda102py38h9f8c3ab_1    conda-forge
pytorch-gpu             1.7.1           cuda102py38hf05f184_1    conda-forge
pyyaml                  6.0             py38h7f8727e_1
readline                8.2             h5eee18b_0
regex                   2022.7.9        py38h5eee18b_0
requests                2.28.1          py38h06a4308_0
sacremoses              0.0.43          pyhd3eb1b0_0
setuptools              65.5.0          py38h06a4308_0
six                     1.16.0          pyhd3eb1b0_1
sqlite                  3.39.3          h5082296_0
tk                      8.6.12          h1ccaba5_0
tokenizers              0.11.4          py38h3dcd8bd_1
tqdm                    4.64.1          py38h06a4308_0
transformers            4.18.0          py38h06a4308_0
typing-extensions       4.3.0           py38h06a4308_0
typing_extensions       4.3.0           py38h06a4308_0
urllib3                 1.26.12         py38h06a4308_0
wheel                   0.37.1          pyhd3eb1b0_0
xz                      5.2.6           h5eee18b_0
yaml                    0.2.5           h7b6447c_0
zlib                    1.2.13          h5eee18b_0
```

CONCLUSION:

Using the trained GPT2 model, we produced 1,20,000 recipes in total. The RecipeDB data was first processed, then special tokens were added, and finally the train set and test set were separated. The GPT2 model, which had been trained on the train set, was fed a list of random ingredients, and it then developed a recipe using those items plus a few extras.

REFERENCES

[1] H. H. Lee, K. Shu, P. Achananuparp, P. K. Prasetyo, Y. Liu, E.-P. Lim, and L. R. Varshney, “Recipegpt: Generative pre-training based cooking recipe generation and evaluation system,” in Companion Proceedings of the Web Conference 2020, 2020, pp. 181–184.
[2] Z. Yu, H. Zang, and X. Wan, “Routing enforced generative model for recipe generation,” in Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2020, pp. 3797–3806.
[3] M. R. Parvez, S. Chakraborty, B. Ray, and K.-W. Chang, “Building language models for text with named entities,” arXiv preprint arXiv:1805.04836, 2018.
[4] C. Kiddon, L. Zettlemoyer, and Y. Choi, “Globally coherent text generation with neural checklist models,” in Proceedings of the 2016 conference on empirical methods in natural language processing, 2016, pp. 329–339.
[5] Y. Agarwal, D. Batra, and G. Bagler, “Building hierarchically disentangled language models for text generation with named entities,” in Proceedings of the 28th International Conference on Computational Linguistics, 2020, pp. 26–38.
[6] R. Koncel-Kedziorski, D. Bekal, Y. Luan, M. Lapata, and H. Hajishirzi, “Text generation from knowledge graphs with graph transformers,” arXiv preprint arXiv:1904.02342, 2019.
[7] Y.-C. Chen, Z. Gan, Y. Cheng, J. Liu, and J. Liu, “Distilling knowledge learned in bert for text generation,” arXiv preprint arXiv:1911.03829, 2019.
[8] B. P. Majumder, S. Li, J. Ni, and J. McAuley, “Generating personalized recipes from historical user preferences,” in Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP). Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 5976–5982. [Online]. Available: https://aclanthology.org/D19-1613
[9] A. Bosselut, A. Celikyilmaz, X. He, J. Gao, P.-S. Huang, and Y. Choi, “Discourse-aware neural rewards for coherent text generation,” arXiv preprint arXiv:1805.03766, 2018.
[10] D. Batra, N. Diwan, U. Upadhyay, J. S. Kalra, T. Sharma, A. K. Sharma, D. Khanna, J. S. Marwah, S. Kalathil, N. Singh et al., “Recipedb: A resource for exploring recipes,” Database, vol. 2020, 2020.
[11] A. Salvador, M. Drozdzal, X. Giro-i Nieto, and A. Romero, “Inverse cooking: Recipe generation from food images,” in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 10 453–10 462.
[12] H. Wang, G. Lin, S. C. Hoi, and C. Miao, “Structure-aware generation network for recipe generation from images,” in European Conference on Computer Vision. Springer, 2020, pp. 359–374.

[13] M. Liang and X. Hu, “Recurrent convolutional neural network for object recognition,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 3367–3375.
[14] R. Dale, “Generating recipes: An overview of epicure,” Current research in natural language generation, pp. 229–255, 1990.
[15] J. Marin, A. Biswas, F. Ofli, N. Hynes, A. Salvador, Y. Aytar, I. Weber, and A. Torralba, “Recipe1m+: A dataset for learning cross-modal embeddings for cooking recipes and food images,” IEEE transactions on pattern analysis and machine intelligence, vol. 43, no. 1, pp. 187–203, 2019.
[16] N. S. Keskar, B. McCann, L. R. Varshney, C. Xiong, and R. Socher, “Ctrl: A conditional transformer language model for controllable generation,” arXiv preprint arXiv:1909.05858, 2019.
[17] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, “Exploring the limits of transfer learning with a unified text-to-text transformer,” arXiv preprint arXiv:1910.10683, 2019.
[18] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, “Language models are unsupervised multitask learners,” OpenAI blog, vol. 1, no. 8, p. 9, 2019.
[19] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” arXiv preprint arXiv:1810.04805, 2018.
[20] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, “Roberta: A robustly optimized bert pretraining approach,” arXiv preprint arXiv:1907.11692, 2019.
[21] K. Greff, R. K. Srivastava, J. Koutn´
ık, B. R. Steunebrink, and J. Schmidhuber, “Lstm: A search space odyssey,” IEEE transactions on neural networks and learning systems, vol. 28, no. 10, pp. 2222–2232, 2016.
[22] Y. Zhang, S. Sun, M. Galley, Y.-C. Chen, C. Brockett, X. Gao, J. Gao, J. Liu, and B. Dolan, “Dialogpt: Large-scale generative pre-training for conversational response generation,” arXiv preprint arXiv:1911.00536, 2019.
[23] N. Garg, A. Sethupathy, R. Tuwani, R. Nk, S. Dokania, A. Iyer, A. Gupta, S. Agrawal, N. Singh, S. Shukla et al., “Flavordb: a database of flavor molecules,” Nucleic acids research, vol. 46, no. D1, pp. D1210–D1216, 2018.
[24] M. Bie ´
n, M. Gilski, M. Maciejewska, W. Taisner, D. Wisniewski, and A. Lawrynowicz, “Recipenlg: A cooking recipes dataset for semi-structured text generation,” in Proceedings of the 13th International Conference on Natural Language Generation, 2020, pp. 22–28.