WEBCRAFT : WHERE WORDS CREATE WEBPAGES

Mukesh Javvaji Sayam Khatri Divya Sri Bandaru Satvika Eda

1. ABSTRACT

The project seeks to construct technology that will let users input text to generate web pages. This project makes use of advances in Natural Language Processing (NLP) and Web Development to create webpages easily. The project's main concepts and objectives include creating aesthetically pleasing web layouts, putting strong natural language processing (NLP) algorithms into practice by reliably interpreting user input, and efficiently organizing the generated content. By integrating the elements of NLP and Web Development, the project aims to provide users with a seamless and intuitive interface for sharing their ideas and stories online. It removes the need for complex coding, allowing users from a wide range of backgrounds to participate in the creation of web pages. In conclusion, this project provides an approachable way to create webpages, addressing a critical need in the digital world. Its creative methodology, which makes use of NLP and Web Development techniques, holds the potential to completely transform the production and dissemination of web content.

2. METHODOLOGY

2.1. NLP Techniques:

- a) <u>LoRA (Low-Rank Adaptation)</u>: It is an efficient fine-tuning technique for fine-tuning large language models. It trains small low-rank matrices instead of the entire model, reducing computational costs and memory requirements. LoRA achieved comparable performance to full fine-tuning while enabling efficient domain adaptation of the large language model to our specific HTML code generation task.
- b) <u>Causal LM</u>: We have used the approach of casual language modelling, a self-supervised model. The model is trained to predict the next token in a sequence based on the previous tokens, without considering any future tokens. By leveraging the learned

patterns and dependencies from large-scale pre-training on huge amount of code data, this became key factor in generating high quality and relevant code outputs.

- c) <u>Byte-Pair Encoding (BPE) Tokenization</u>: Code Llama uses the same BPE tokenizer as Llama 2, which is effective for handling rare and outof-vocabulary words in code by breaking them down into smaller, meaningful sub word units.
- d) <u>Padding and Truncation</u>: Preprocessing technique used to handle variable-length sequences. Padding and truncation help ensure that all input sequences have the same length by either adding special tokens (padding) to shorter sequences or cutting off (truncating) longer sequences to a desired maximum length.



Fig 1: Model Architecture

3.	EXPERIMENTAL	SETUP	INCLUDING
	HYPERPARAMETER		CHOICES

3.1. Data Collection:

The first step of our project involved acquiring a dataset consisting of 1.5 million URLs, each assigned with one of the 15 available categories. These URLs are then scraped using HTTP requests to get the HTML code of the webpages. This served as the initial step in our attempt to collect the data, enabling us to retrieve web data of distinct categories.

This collection of HTML data from diverse URLs representative of several different categories has laid the groundwork for our project. This enabled us to proceed with the subsequent phase - preprocessing.

3.2. Preprocessing:

In the data processing phase, we employed an HTML parser, namely BeautifulSoup, to extract structured information from the responses gathered from HTTP requests. To ensure the quality and relevance of the data, a filtering mechanism was implemented whereby webpages containing less than 10,000 characters were disregarded. This threshold was set to exclude pages with potentially insufficient content or error messages such as "PAGE NOT FOUND," thereby providing meaningful information in the data.

Furthermore, as our project primarily focused on HTML and CSS, additional preprocessing steps were conducted to refine the parsed responses. Specifically, JavaScript code, Meta tags, and comments from the extracted HTML content were removed. The decision to exclude JavaScript code aimed to isolate the structural and stylistic aspects of webpages from dynamic behaviours functionalities introduced and through JavaScript. Additionally, eliminating Meta tags and comments helped streamline the data by reducing extraneous information, ensuring that the data remained centred on core HTML structure and CSS styling elements.

3.3. Prompt Generation:

In In accordance with the methodology outlined in section 2.1 of the project, each URL was associated with a specific category. To maintain ethical standards, webpages categorized as "Adult" were excluded during the web scraping process. Subsequently, to generate prompts for our task, we adhered to a standardized format. This format involved creating prompts structured as follows: "Create a {Category} web page for {title}." For instance, an example prompt generated using this format could be: "Create a Health website for Dalmane (Flurazepam): Side Effects, Uses, Dosage." By employing this format for prompt generation, each prompt accurately reflected the category and HTML content of the associated webpage. These prompts were then saved in JSON format for fine tuning the LLM's. Ultimately, this approach yielded approximately 100,000 prompts, each paired with its corresponding HTML, forming a comprehensive dataset for finetuning.

3.4. Model Fine-tuning:

<u>GPT-2</u>:

The pre-trained GPT-2 tokenizer (GPT2Tokenizer) and model (GPT2LMHeadModel) are imported from the Hugging Face Transformers library. The dataset was loaded and batched using a custom HTML Dataset class and a DataLoader object. We have used the collate_fn method, a custom function used by PyTorch's DataLoader to collate a batch of data samples into a format suitable for the model.

During training, the model is supposed to generate HTML code from natural language prompts by minimizing the loss between the model's predictions and the target HTML outputs. We have defined a AdamW optimizer with a learning rate of 0.01 to update the model parameters during training to improve convergence and generalization. The total number of training steps was calculated by multiplying the length of the dataloader (number of batches) by the number of epochs. By saving the fine-tuned model and the entire model object, you can use the trained model for inference or continue training from the saved checkpoint.

Falcon:

Using the pretrained "falcon-7b-sharded-bf16" from Hugging Face and optimized the efficiency through 8-bit quantization. Loaded a tokenizer from Falcon which converts text into tokens and adds padding.

JSON files containing training and testing data are loaded and split, with relevant data formatted into DatasetDict for efficient handling. The training process itself is orchestrated using the "SFTTrainer" class, which is designed for sequence-to-sequence models. This class utilizes TrainingArguments to define detailed training parameters like batch size, learning rate, optimizer settings, and a learning rate schedule. The model is configured to use LoRA (Low-Rank

Adaptation) to modify only certain layers of the model efficiently, which is especially useful for adapting large pre-trained models to new tasks with minimal computational overhead. Finally, the training process includes callbacks for early stopping to prevent overfitting and a detailed setup for handling various data and model configurations during the training iterations.

After training, the model is saved along with its tokenizer. It is re-trained over various batches of datasets to generate HTML based on new prompts.

Llama-2:

The fine-tuning process involved initializing the "Ilama-2-7b-hf" model, which is available in the Hugging Face, with pre-trained weights and fine-tuning it on the curated dataset. We have utilized the LoRA configuration to reduce the number of training parameters.

SFTTrainer, a specialized trainer for structured data generation tasks, was utilized to streamline the fine-tuning process. The fine-tuning process employed a Causal Language Model (LM) setup, where the model predicts the next token in a sequence based only on the preceding tokens. The fine-tuning procedure utilized the Adam optimizer, a popular choice for fine-tuning Large Language models.

After fine tuning, the model was able to generate some markup, but the code cannot be rendered as an HTML webpage.

Code Llama:

Using the pretrained "codellama/CodeLlama-7bhf" from Hugging Face and optimized the efficiency through 8-bit quantization. Loaded a tokenizer from Code Llama which converts text into tokens and adds padding.

Before the actual training begins, the model is configured with LoRA, a technique that adds trainable low-rank matrices to modify specific weights in the model's layers. We use "Trainer" object from the Hugging Face to orchestrate the training process by managing batch size, gradient accumulation, learning rate etc. The Trainer starts the training process, which involves multiple iterations over the training dataset. Over various iterations, the model makes predictions based on the input data, compares these predictions to the actual data using Cross Entropy Loss function, and updates its weights to reduce prediction errors.

After training, the model is saved along with its tokenizer. It is re-trained over various batches of datasets to generate HTML based on new prompts.

3.5. Hyperparameter choices

The following are the hyperparameters used for fine-tuning the model:

- 1. <u>Epochs</u>: The training process was conducted over 3 epochs.
- 2. <u>Learning Rate:</u> The learning rate was set to 0.002.
- 3. <u>LoRA Alpha</u>: The alpha value affects the strength of the regularization applied during training.
- 4. <u>LoRA Dropout</u>: A Dropout rate of 0.1 was utilized for LoRA.
- 5. <u>LoRA Rank</u>: The LoRA Rank parameter was set to 64. This parameter determines the rank of the approximation matrix used in LoRA regularization.

These hyperparameters were carefully selected based on experimentation to optimize the finetuning process of the LLM models. The values mentioned above strike a balance between model performance, training efficiency and computational resources available.

4. **RESULTS**

Initially, we fine-tuned the GPT2 model to train on our dataset but GPT2 is taking a lot of time to even train a small set of data from the entire dataset. It is having a maximum sequence length of 1024, but the HTML in training data has more than 10000 characters. So, we had to chunk each training html code to the maximum length of 1024, making it difficult for the model to interpret the training data.

The Falcon-7B-Sharded-BF16 failed to produce valid HTML code but it generated some text data for the given prompt.

create a Health website for Dalmane (Flurazepam): Side	Effects, Uses, Dosage, I	nteractions, Warni	ngs & Precaution
Dalmane (Flurazepam) is a benzodiazepine used to treat	insomnia. It is availabl	e in generic form.	Common side eff
Dalmane (Flurazepam) is a benzodiazepine used to treat acts include drowsiness dizziness and headache	insomnia. It is availabl	e in generic form.	Common side eff
Dalmane (flurazepam) is a benzodiazepine used to treat	insomnia. It is availabl	e in generic form.	Common side eff
Dalmane (flurazepam) is a benzodiazepine used to treat	insomnia. It is availabl	e in generic form.	Common side eff
Dalmane (flurazepam) is a benzodiazepine used to treat	insomnia. It is availabl	e in generic form.	Common side eff
Dalmane (flurazepam) is a benzodiazepine used to treat	insomnia. It is availabl	e in generic form.	Common side eff
Dalmane (flurazepam) is a benzodiazepine used to treat	insomnia. It is availabl	e in generic form.	Common side eff
Dalmane (flurazepam) is a benzodiazepine used to treat	insomnia. It is availabl	e in generic form.	Common side eff
ects include drowsiness, dizziness, and headache. Dalmane (flurazepam) is a benzodiazepine used to treat	insomnia. It is availabl	e in generic form.	Common side eff
ects include drowsiness, dizziness, and headache. Dalmane (flurazepam) is a benzodiazepine used to treat	insomnia. It is availabl	e in generic form.	Common side eff
ects include drowsiness, dizziness, and headache. Dalmane (flurazepam) is a benzodiazepine used to treat	insomnia. It is availabl	e in generic form.	Common side eff
ects include drowsiness, dizziness, and headache. Dalmane (flurazepam) is a benzodiazepine used to treat	insomnia. It is availabl	e in generic form.	Common side eff
ects include drowsiness, dizziness, and headache. Dalmane (flurazepam) is a benzodiazepine used to treat	insomnia. It is availabl	e in generic form.	Common side eff
ects include drowsiness, dizziness, and headache. Dalmane (flurazepam) is a benzodiazepine used to treat	insomnia. It is availabl	e in generic form.	Common side eff
ects include drowsiness, dizziness, and headache. Dalmane (flurazepam) is a benzodiazepine used to treat	insomnia. It is availabl	e in generic form.	Common side eff
ects include drowsiness, dizziness, and headache. Dalmane (flurazepam) is a benzodiazepine used to treat	insomnia. It is availabl	e in generic form.	Common side eff
ects include drowsiness, dizziness, and headache.			
Fig 2:	Fine-tuned	Falcon	output
0			

We have fine-tuned the Llama2-7B-HF model for a set of datasets and was able to produce some text data along with the empty markup tags.

### Input: Create a health website for Self-Improvement strategies for mental health>	
### Response:	
! <create a="" for="" health="" mental="" self-improvement="" strategies="" website=""> >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>></create>	
Fig 3: Fine-tuned Llama-2 output	

Parallelly fine-tuning efforts led to the successful operation of CodeLlama-7B-HF, which effectively generated HTML tags and text properly.

### Input:	
Create a Health website for Dalmane (Flurazepam): Side Effects, Uses, Dosage, Interactions, Wa	rnings.
### Response:	
html	
<html></html>	
<head></head>	
<title>Dalmane (Flurazepam)</title>	
<style></style>	

Fig 4: Fine-tuned CodeLlama output

Further training on the already trained model of CodeLlama, we were able to produce a complete static html page which can be rendered on the browser without extra unnecessary data in the HTML.

Input:
Create a Health website for Self-Improvement Strategies for Mental Health.
Response:
html
<html></html>
<head></head>
<title>Health website for Self-Improvement Strategies for Mental Health</title>
<body></body>
<h1>Self-Improvement Strategies for Mental Health</h1>
Explore Self-Improvement Strategies for Mental Health with our Health website.
Here, you can learn about:

Meditation Techniques
Self-Hypnosis
Stress Management
Fig 5: Code generated by fine-tuned Code Llama model

Self-Improvement Strategies for Mental Health

Explore Self-Improvement Strategies for Mental Health with our Health website. Here, you can learn about:

- · Meditation Techniques
- Self Hypnosis
- Stress Management

Fig 6: Web display of the code generated (code in fig 5)

With further training, the model was able to generate a web page which is a gallery of photos. The model added random images to the web page i.e. generated the img tags efficiently.

Input:
create a Recreation website for Zanzig.com Photo Hub - Photos From The Pas
Response:
html
<html></html>
<head></head>
<title>Zanzig.com Photo Hub - Photos From The Past</title>
bodyf
background-color: #E5E5EA:
}
.container{
max-width: 1200px:
margin: 0 auto:
padding: 20px:
}
.header{
text-align: center:
margin-bottom: 20px:
}
.photo{
display: inline-block;
width: 30%;
margin-bottom: 20px;
}
1mg{
width: 100%;
height: auto;
}
<body></body>
<div class="container"></div>
<div class="header"></div>
<h1>Zanzig.com Photo Hub - Photos From The Past</h1>
Prowse through our selection of classic photos from the past.





Fig 8: Web display of the code generated (code in fig 7)

Below are the results for the training loss on the final fine-tuned code llama model.



Fig 9: Training loss of fine-tuned Code Llama





Fig 10: Validation loss of fine-tuned Code Llama

5. CONCLUSION

Fine tuning the models based on generic LLM's that are not specific to code generation have produced intermediary results. On the other hand, fine tuning CodeLlama has given optimal results over all other LLM's with generation of both HTML and CSS along with images with less training and validation loss. With larger datasets, we would be able to produce more aesthetically pleasing webpages that are dynamic in nature.

There were a couple of limitations observed such as the availability of computational resources which posed a substantial constraint, influencing both the complexity and scale of the project. The content generated by the model can face potential copyright infringement concerns, necessitating careful attention to the sourcing of materials.

5.1. Future Focus

There are several opportunities for improvement in future coursework.

- Addressing the addition of JavaScript functionality would enhance the project's usefulness and user engagement.
- Enabling multi-webpage generation would expand the project's capabilities and allow for the creation of more extensive websites with interconnected content.

In conclusion, while the project encountered several limitations, there are many opportunities for growth and improvement in future coursework. By addressing these limitations and focusing on enhancing functionality, design, and usability, future projects can strive to deliver more compelling and impactful web experiences.

6. GITHUB REPOSITORY

GitHub: <u>https://github.com/satvika-</u> eda/WebsiteGenerationNLP.git

7. TEAM CONTRIBUTIONS

The team collectively managed dataset exploration, acquisition, and preprocessing. Individual contributions include:

- Sayam Khatri: Fine-tuning and training of the Falcon model.

- Divya Sri Bandaru: Fine-tuning and training of the GPT2 model.
- Mukesh Javvaji: Fine-tuning and training of the Llama2 model.
- Satvika Eda: Fine-tuning and training of the Code-LLama model.

8. REFERENCES

- i. Calò, T. and De Russis, L., 2023, May. Leveraging Large Language Models for End-User Website Generation. In International Symposium on End User Development (pp. 52-61). Cham: Springer Nature Switzerland.
- ii. PyTorch. (n.d.). torch.nn.Transformer. PyTorch Documentation. Retrieved from <u>https://pytorch.org/docs/stable/generat</u> <u>ed/torch.nn.Transformer.html</u>
- Hugging Face. (n.d.). Soft Trainers for TRL. Hugging Face Documentation. Retrieved from <u>https://huggingface.co/docs/trl/v0.4.7/e</u> n/sft trainer
- iv. Hugging Face. (n.d.). Posterior Estimation for Fine-Tuning. Hugging Face Documentation. Retrieved from <u>https://huggingface.co/docs/peft/en/ind</u> <u>ex</u>
- v. Hugging Face. (n.d.). Quantization. Hugging Face Documentation. Retrieved from <u>https://huggingface.co/docs/accelerate/</u> en/usage guides/quantization
- vi. Hugging Face. (n.d.). LoRA. Hugging Face Documentation. Retrieved from https://huggingface.co/docs/peft/en/pac kage reference/lora
- vii. Hugging Face. (2020, July 6). GPT2. Hugging Face Transformers Documentation. Retrieved from <u>https://huggingface.co/transformers/v3.</u> 0.2/model_doc/gpt2.html