Reverse Browser: Image-to-Code Generator Using Transformers

Zoltán Tóth-Czifra

Kellogg College

University of Oxford

A thesis submitted for the degree of

Master of Science (MSc) in Software Engineering

Trinity Term 2025

Supervised by Max Van Kleek

Abstract

Automating the task of turning user interface design into code (image-to-code or image-to-UI) is an active software engineering research area. However, the state-of-the-art solutions do not achieve high fidelity to the original design, as evidenced by benchmarks [1, 2].

In this work, I attempt to cut the Gordian Knot by approaching the problem differently: I use vector images instead of bitmaps as model input. I review related works and discuss their contributions and shortcomings. I create several large cardinality datasets for training machine learning models, both synthetic and scraped from the public web. I evaluate the available array of Image Quality Assessment (IQA) algorithms and introduce a new, multi-scale metric. I then proceed to model selection and discuss the merits and drawbacks of different transformer-based model architectures. Using the selected models, I conduct a series of training experiments, first validating my approach on simple synthetic data, then training large models on complex data collected from the public web. Finally, I evaluate the most capable trained model and discuss its limitations.

Additionally, I release the datasets, the implementation of the new IQA metric, the trained model, and the code and metrics of the experiments under an open license.

Acknowledgements

I would like to thank my supervisor, Max Van Kleek, for providing feedback on this work.

In addition, I would like to thank my friends Juxhin, Marion, and Shaaz for their advice and emotional support.

Lastly, I wish to express immense gratitude to the unpaid, passion-driven contributors of the free and open-source software that made this project possible (see Appendix A.7 for a list).

Contents

Abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
CNN	Convolutional Neural Network
CSS	Cascading Style Sheets
CUDA	Compute Unified Device Architecture
FLOP	Floating Point Operation
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HTML	Hypertext Markup Language
IQA	Image Quality Assessment
LLM	Large Language Model
LPIPS	Learned Perceptual Image Patch Similarity
PoC	Proof of Concept
SVG	Scalable Vector Graphics
UI	User Interface

k, M, B, T	thousand, million, billion, trillion

1 Introduction

1.1 Motivation

The World Wide Web is by far the most popular software development platform. A 2024 survey of over 65,000 developers [3] indicated that the top two languages used by professional software developers were JavaScript (62.3%) and HTML+CSS (52.9%). The latter is especially striking because HTML and CSS are not general-purpose programming languages, but markup languages used to build user interfaces for the web. On Stack Overflow[1], a popular software development Q&A site, HTML ranks as the 8th most popular tag for all questions, while CSS ranks 13th [4].

While building web interfaces is a visual task by nature, software developers spend considerable time implementing visual features through code, specifically HTML and CSS, as shown by the survey. In a 2022 questionnaire [5] 37.91% of the respondents indicated that they dread working with HTML+CSS.

A typical HTML+CSS task is called site-building. The software engineer takes a set of images prepared by a user interface (UI) designer and implements a high-fidelity version of these designs in frontend code[2]. Is the manual, labor-intensive task of turning these designs into HTML+CSS necessary? Can it be automated? The user interface designer has performed the creative work. Implementing the designs in a format that a browser can render requires much less creative input.

Automating code generation from UI design would enable software engineers to prototype quickly and designers to implement their creations without coding knowledge. Such a tool

[1] https://stackoverflow.com/

[2] Frontend code refers to client code typically running in a web browser, consisting of HTML, CSS, and JavaScript.

would eliminate tedious and dreaded software engineering tasks and let software engineers focus their efforts on areas where they can add more value.

Recent advances in deep learning, particularly the transformer architecture [6] and its derivations such as Large Language Models, have enabled rapid progress in powerful Artificial Intelligence (AI) models. One of these models' most exciting applications is coding assistance [7]. Coding assistants are generative tools that act as a conduit between the software engineer's intent and the resulting code, reducing syntax-heavy keystrokes and speeding up coding tasks. They are imperfect, but they have already gained mass popularity. According to a 2024 GitHub survey conducted among software engineers, "97% of respondents reported having used AI coding tools" [8]. I believe the AI-assisted coding tool chest is incomplete without a tool that turns visual input into code.

My approach differs from prior work because it uses vector images as model input instead of bitmaps. Vector images may be more appropriate for image-to-code tasks because they generally have lower intrinsic dimensionality than a bitmap of the same design and contain explicit structural information. This information may be much more naturally translatable to another document structure than the implicit, hard-to-extract structure embedded in a bitmap.

The most popular design software used by UI designers [9], like Sketch[3], Figma[4], and Adobe Illustrator[5], can export designs as vector graphics. Exporting to a bitmap causes loss of explicit structural information.

1.1.1 Contributions

In this project, I set out to create a model capable of producing web code from vector-based web designs.

In that process, I designed and implemented methods to produce large-scale training datasets from synthetic data as well as the public web. I discussed the main features of the crawling software and my design decisions. I demonstrated different strategies for producing synthetic data. I published the crawling software, five large synthetic training datasets, and three large public web training datasets.

I evaluated several different Image Quality Assessment (IQA) algorithms to measure the accuracy of the data collection and the trained models. I developed and released a new IQA metric called Multi-Scale Pixel Similarity (MSPS).

I assessed several candidate model architectures to train on before settling on two model families. I performed 45 documented training experiments, made the code and training metrics available, and analyzed several of these in detail.

[3]https://www.sketch.com/
[4]https://www.figma.com/
[5]https://www.adobe.com/products/illustrator.html

Furthermore, I released the weights of the best-performing final model and analyzed its performance. I wrote more broadly on the prospects of such a model and the ethical implications of its use. Additionally, I published the computing cost calculations relating to this project.

All the released software, datasets, and models are available under the Apache 2.0 license.

From a wider perspective, the project can be thought of as a case study of a student working on modern, powerful AI models.

Producing these large models often requires computing resources beyond the reach of an average university student. This project demonstrates the limitations imposed by those resource requirements. It makes the point that software engineering students are at risk of falling behind on some of the most important and powerful computing technologies due to the costs of getting deeply acquainted with them.

2 Background

2.1 Related technologies

In this section, I provide a brief context relating to the technologies touched on in this project.

Web In the context of this project, (World Wide) Web refers to the application platform rather than the information sharing system. It is a collection of technologies, principally the Hypertext Transfer Protocol (HTTP), the Hypertext Markup Language (HTML), Cascading Style Sheets (CSS), and JavaScript. In this project, I will focus on HTML and CSS, which are responsible for the visual presentation of web applications. I will often collectively refer to them as *markup* for brevity, although the word's traditional meaning refers only to HTML.

Web browsers and rendering engines A web browser is a software application that lets end users access web pages and web applications. The rendering engine is a browser subcomponent that takes HTML, CSS code, and other resources (images, videos, etc) and converts them into an interactive image, ultimately a set of pixels that may respond to user interaction.

Responsive web design Traditional web design focused on a single device type and therefore screen resolution, mainly desktop computers. An extension of CSS called *media queries* [10, 7 Media types] lets the web design dynamically adapt to different screen resolutions and device types (e.g., mobile phones, tablets), delivering the same frontend code for all devices.

Vector graphics Computer graphics generally fit into two categories: bitmaps and vector graphics. A bitmap is a two-dimensional grid of pixel color values. A vector image, in contrast, is constructed using mathematical descriptions of shapes (paths, polygons, curves) as well as embedded text and images. Popular formats include Scalable Vector Graphics (SVG) [11], Encapsulated PostScript (EPS), and Adobe Illustrator format (AI). Most of the popular user interface design programs can export designs as vector graphics.

Transformers and Large Language Models A transformer is an artificial neural network architecture introduced by Vaswani et al. [6]. It improves on previous state-of-the-art models like Recurrent Neural Networks (RNN) and Long Short-term Memory (LSTM) by replacing recurrence with an attention mechanism. The self-attention mechanism in transformers can capture long-range dependencies between input and output sequences. Principally proposed for machine translation tasks, the transformer architecture succeeded in verious other domains.

Perhaps most prominently, it is the underlying architecture of modern Large Language Models (LLMs). These models are trained on large amounts of text (natural language and sometimes computer code) to perform language modeling tasks, predominantly text generation, although some LLMs are also used for classification, text embedding, sentiment analysis, etc. The original transformer architecture proposed a two-legged design: a text encoder and a decoder, connecting to the former through the cross-attention mechanism. The latest, most powerful LLMs go beyond the encoder-decoder model and employ a decoder-only architecture. Section 6.1 has more details.

Large Language Models are auto-regressive because they predict only the next token at a time. Some modern LLMs are multi-modal in that they can ingest images, audio, or video, not just text. The impressive abilities of LLMs prompted many commercial and open-source products, such as ChatGPT[1], Gemini[2], Mistral[3], Llama [12], or DeepSeek R1 [13].

Tokenizer Typically, transformer models do not directly take text as input. First, the input is *tokenized*, in other words, parts of the text (typically words or subword sequences) are mapped to a numeric dictionary. This shortens the sequence, which is important for performance optimization for the quadratic attention mechanism, trading it for a larger "vocabulary". The model's output also consists of tokens and is decoded by the same tokenizer.

[1] https://chatgpt.com/
[2] https://gemini.google.com/app
[3] https://mistral.ai/

2.2 Related work

Turning images into UI code using machine learning techniques is an active area of research. However, none of my prior works used vector images as input.

Pix2code [14] uses LSTM with convolutions to turn an input bitmap into HTML+CSS markup and graphical user interface (GUI) code for other platforms. It was trained on relatively small cardinality synthetic training data of randomly generated GUI screenshots and a corresponding, simplified domain-specific language (DSL) that describes the GUI. The DSL emitted by the trained model can be converted into corresponding HTML+CSS code, iOS, or Android code. According to the author, the results are of low quality due to the model's small parameter space. The simple DSL further limits the practical application.

A 2025 paper introduces **Frontend Diffusion** [15], which works similarly. It builds a DSL document from hand-drawn sketches. It goes further and translates them to SVG vector images. However, interestingly, it does not use the SVGs directly. Instead, it converts them into bitmaps and uses them as input for a commercial Large Language Model (Claude 3.2 Sonnet) to generate frontend code. The tool is not meant for precise design implementation, only rough prototyping.

Sketch2code [16] is another model that turns bitmaps of hand-drawn wireframes into markup. The authors contrast classical machine learning and computer vision methods with deep learning. For the deep-learning model, they employ convolutional neural networks (CNN) to recognize a minimal set of drawn element types. Using the approximate position of these elements, a simple HTML+CSS page is then generated. The resulting model is limited in terms of the design and the complexity of the output.

Pix2Struct [17] is related but sets a different goal. It is a Visual Document Understanding model built to recognize the structure of documents such as receipts and business cards. It is an improvement over the previous state-of-the-art model, Donut [18]. It uses a multimodal transformer trained on variable-resolution bitmap inputs and masked HTML+CSS counterparts. The training set consists of a large number of real-world websites, and the model has an ample parameter space. Pix2Struct does not generate HTML+CSS but predicts the visual structure of a document from an image.

ViCT [19] aims to turn web page screenshots or designs into HTML+CSS code. Similarly to Pix2Struct, it uses a multimodal transformer where the encoder leg is either a Vision Transformer (ViT) or DiT (Document Image Transformer), and the decoder leg is a text-based GPT-2 [20] or Llama [12] model. It introduces relevant ideas that I have built on here.

It uses synthetic input data from simple, randomly generated websites and screenshot counterparts. It fine-tunes the model using reinforcement learning, where a similarity

score between the original screenshot and the one from the resulting webpage is used as input. Unlike Pix2Struct, its training data is simplistic, with a maximum of six HTML elements. As the input gets more complex, the performance scores quickly deteriorate. The authors note the model's limitations and admit that their training data "may not fully encapsulate the complexity of real-world web pages."

A very recent paper published on 22 April 2025 introduces **UICopilot** [21]. The authors recognize the complexity of real-world web designs that previous works failed to address. They take design bitmaps as input and first extract the high-level document structure using a custom-trained vision transformer. They then use the extracted structure and the original bitmap as input for a commercial LLM (GPT-4 [22]) to produce the corresponding UI code. The paper validates my approach of extracting structural information from webpages using the rendering engine of a web browser and using the obtained data for model training. It differs significantly from it in that it relies on a bitmap input along with the structural data. My approach, in contrast, retains the extracted structure implicit in a vector file.

A recently proposed benchmark from Stanford University, **Image2Struct** [1] aims to track the progress of the image-to-code capabilities of vision-language models. They write: "we find that VLMs are unable to perform well on our tasks", indicating that further research is necessary.

Some commercial tools exist that generate frontend code from images.

Uizard's "Screenshot Scanner"[4] tool implements a bitmap-to-markup machine learning model. In my tests, the results were low-fidelity.

Other tools, such as Replit[5], GitHub Codespaces[6], and Firebase Studio[7], attempt to automate the entire software engineering process, including building UI. However, their primary modality is text, which is unlikely to produce high-fidelity design implementations. The Starry Night would surely be a less renowned painting if Vincent Van Gogh had made it by instructing an apprentice holding the paintbrush. In other words, design is a visual art, not textual.

Some of these models are multi-modal and support bitmap inputs. However, the output fidelity was very low in my tests, or the tool simply did not work.

[4]https://uizard.io/screenshot-scanner/
[5]https://replit.com/
[6]https://github.com/codespaces
[7]https://studio.firebase.google.com/

3 Goals

The principal goal of the project is stated as follows:

PG *Produce a machine learning model that takes vector image design file(s) as input and produces a corresponding, high-fidelity web page code (HTML and CSS) automatically.*

All other objectives are subordinate to this main goal. From the project's conception, it was clear that the requirements could be organized into three major interdependent areas of work.

Throughout the main text, I will use the markers such as these to indicate content related to particular goals and requirements, respectively: DG2 MR3

3.1 Data

These sub-goals and corresponding requirements are born from the simple recognition that training machine learning models requires large amounts of data.

Sub-goals

- DG1 Produce several simple, synthetic datasets of web pages and their corresponding vector images that help prove that models can learn simple web features TG2
- DG2 Create a large dataset of complex, diverse examples of web pages and corresponding vector images to train the final model TG3

Functional requirements

- DR1 **Format** A widely used vector image format should be used that popular design software can export to
- DR2 **Fidelity** The pairs of vector images and rendered web pages should match visually with high fidelity MG1 MG2
- DR3 **Quality** The data should be clean of low-quality examples that do not contribute to learning MG2
- DR4 **Item size** Individual data items should be small enough to fit the chosen model architecture's input limits TG1 TF4
- DR5 **Total size** The datasets should have a large enough cardinality to conceivably produce the desired model TF3
- DR6 **Non-interactive** The web page code should be clean of interactive or media elements that would pollute the data set
- DR7 **Responsiveness** The vector image should be collected in several different popular screen resolutions so that the model can produce a responsive web design TR2

Non-functional requirements

- DR8 **Performance** Data production and collection should be fast enough to complete within a few weeks
- DR9 **Legality and ethics** Where the public web is used as a source, data should be collected lawfully and ethically, respecting web crawler conventions and protocols
- DR10 **Transparency** The compiled datasets should be made available under a permissive license

3.2 Metrics

The collected datasets and the model need reliable metrics to judge the image fidelity and performance.

Sub-goals

- MG1 Define an accuracy metric (or metrics) that can reliably judge pairs of vector images and web pages for their perceptual similarity FD2 TR3
- MG2 Define an accuracy threshold that valid dataset items should meet FD2

Functional requirements

- MR1 **Inputs and output** The metric should take two RGB bitmaps as input and output a single numeric score
- MR2 **Resolution** It should be resolution-independent and work with an extensive array of smaller or larger images
- MR3 **Small translations** It should be robust against minor, few-pixel translation differences between the images
- MR4 **Other transformations** It should be sensitive to other transformations such as rotation, scale, and large translations DR3
- MR5 **Continuity** The metric should gradually change depending on the similarity between images
- MR6 **Perception** The metric should closely match human perception when judging similarity between two images containing web page designs
- MR7 **Automaticity** The metric should be automatic (i.e., not based on human input)

Non-functional requirements

- MR8 **Performance** The metric should calculate in a few seconds and run on commodity hardware (e.g., a regular PC) DR8 TR5

3.3 Model

With the dataset and metrics defined, the model itself is the last piece of the puzzle.

Sub-goals

- TG1 Select a model architecture (or architectures) to train
- TG2 Train the selected model architecture(s) to learn to translate simple web features from design to code (PoC)
- TG3 Train the selected model architecture(s) on the complex dataset

Functional requirements

- TR1 **Inputs and output** The model should take a single SVG file and produce valid HTML+CSS output that can be rendered in a browser
- TR2 **Responsivity** The model may also take several SVG files representing different screen resolutions and produce a single responsive HTML+CSS output that, rendered in those screen resolutions, matches the corresponding design
- TR3 **Accuracy** The model's output, when rendered in a web browser, should closely match the input design(s), ideally down to the pixel level
- TR4 **Size** The model should be able to process complex design inputs of at least several tens of kilobytes

Non-functional requirements

- TR5 **Training performance** The chosen model architecture should be trainable on all of the datasets within a reasonable amount of time (one week maximum) with available and affordable hardware, ideally a single high-end GPU DG1 DG2
- TR6 **Inference performance** The trained model should run on available and affordable hardware and should output a single web design instance in under a minute
- TR7 **Transparency** The trained model should be made available under a permissive license

3.4 Risks

Some of the goals and requirements have a serious risk attached to them and are not guaranteed to succeed.

The main risk is the model's final performance. The nature of large-scale machine learning projects is that the final performance at the end of the training cannot be accurately predicted. The entire objective of TG2 is to reduce this risk as much as possible.

Training large machine learning models on ample training data is expensive. The project's time and financial budgets are limited, and there is a real danger of running out of these before an acceptable model can be trained.

Modern machine learning architectures (such as the *transformer*) are robust but often come with limitations in terms of their resource use or input parameters. The model selection process is not assured to yield an appropriate architecture.

There are risks associated with the training data collection. It is unclear whether high-fidelity vector images can be produced from web pages. How much data is needed to produce a high-quality model is also uncertain. There are open legal questions about the use of public web data.

3.5 Out-of-scope

The following ideas, features, or concepts are out of scope for this project:

- Supporting several vector image inputs
- A user-friendly interface that integrates with the design software and the model
- Weight pruning or other performance optimization of the final model
- The ability to edit an existing HTML+CSS code instead of generating new instances
- Using hints in the design as to what markup element type should be generated (e.g., input boxes, links)
- Emphasis on the code quality of the generated HTML and CSS code

4
Data

Machine learning can be thought of as lossy compression over input data [23]. This suggests that the input quality directly determines the trained model's quality. I spent considerable effort in obtaining high-quality training data. The ultimate objective is to "translate" vector graphics to markup, so I needed to obtain corresponding pairs of these.

Why vector images? In Chapter 1, I already hinted at reasons to use vector images over bitmaps. I will unpack those arguments here.

The degrees of freedom in a bitmap file can be described by the formula 4.0.0.1.

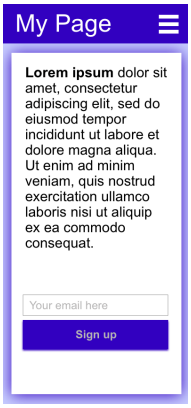
$$D_{\text{bitmap}} = W \times H \times C \tag{4.0.0.1}$$

Here W and H are the width and height of the bitmap and pixels, and C is the number of color channels. The latter is typically 3, while the width and height may be a few hundred to a few thousand for a typical web design. In contrast, the degrees of freedom of a vector image may be described by 4.0.0.2.

$$\begin{aligned} D_{\text{vector}} &= \sum_{i=1}^{N} d_i = N\,\bar{d} \\ d_i &= \#\{\text{numeric parameters of primitive } i\} \\ N &= \#\{\text{primitives}\} \\ \bar{d} &= \text{mean } d_i \end{aligned} \tag{4.0.0.2}$$

A primitive is a mathematically described geometrical shape, such as a curve or a rectangle. For a typical web design, $N \ll 1000$ and $\bar{d} \ll 100$.

The degrees of freedom closely relate to "compressibility." An elementary experiment shown in Table 4.1 of saving a simple design in different formats provides empirical evidence. Intuitively, learning from fewer degrees of freedom is faster and requires smaller models.



Format	File size (bytes)
SVG	4,752
PNG	19,692
JPEG	53,408

Table 4.1: A simple web design (on the left) saved in different image formats. The pixel dimensions are 393×852. SVG is a vector image format, while PNG and JPEG are bitmaps. PNG is a lossless image format, while JPEG is lossy. Here, the JPEG was exported with 80% quality setting. Both bitmaps went through additional optimization using TinyPNG and TinyJPG. The SVG file contains uncompressed text, which can be further reduced to 1,764 bytes by compressing.

Besides the lower dimensionality, a strong argument for using vector images is that they contain explicit structure. The position and size of each primitive and nested group indicate the document's structure directly. The model can access this information without having to first extract spatial structure from a bitmap with a convolutional neural network.

On the decoding side, markup (HTML+CSS) has similar characteristics. A web page's Document Object Model (DOM) is a series of nested primitives and their attributes. My intuition is that it is easier to learn to translate between inputs and outputs of similar quality, rather than mixed modalities.

Additionally, a bitmap of the web design will have all the other bitmaps that appear in the design embedded in it (e.g., a background image or a large header image). A vector file can treat these as external resources and replace them with references. Then, the same references (optionally transformed) can be used in the markup that the model outputs.

My research did not surface any public dataset with pairs of vector images and HTML+CSS, so I decided to create my own. DG1 DG2

As for the vector format, I chose Scalable Vector Graphics (SVG) [11]. DR1 It is an open format with wide support. Other popular but proprietary vector formats, such as Adobe Illustrator Artwork (AI) and CorelDRAW Document (CDR), can be automatically

converted to SVG. This is important if the trained model is expected to be used in a UI designer workflow. Importantly, SVG images can be rendered by modern browser engines, making data crawling and scoring easier.

4.1 Rendering

A browser rendering engine parses HTML and CSS, builds tree structures of both (Document Object Model [DOM] and CSS Object Model [CSSOM]), and computes a layout with the size and position of each page element from these trees. The last step is painting, which converts the layout to pixels to be displayed [24]. The layout, complete with the DOM and CSSOM, remains accessible through the browser's API and can be thought of as a structural representation of the page.

To obtain an SVG-markup[1] pair, the web page first has to be rendered in a browser engine. Then, by iterating over the layout, the vector image can be constructed based on the attributes of each page element, such as position, size, color, text style, etc. This is a task with non-trivial complexity. Difficulty arises from technical differences between HTML and SVG specifications, such as the availability of element types and syntactic, styling, and stacking order differences.

Importantly, the resulting SVG file will represent the web page with a particular viewport size and will not dynamically reflow elements like a web page. Multiple SVG files corresponding to different viewport sizes may be generated.

4.1.1 Methods

The rendering tool I implemented for this task does the following:

1. Loads a **list of URLs** to be rendered. These may be on the public internet or hosted locally in case of synthetic data.
2. Checks whether the website **allows crawling** with the Robots Exclusion Protocol [25]. DR9
3. **Disables script** execution that might change the page's look dynamically. It also disables custom fonts. DR6
4. Makes sure all "lazy-loaded" elements that depend on the user's scroll position are **loaded**.
5. **Saves** the web page to the hard drive to ensure its content is static and self-contained.
6. Sets the **viewport size** to one of the pre-defined sizes
7. **Opens** the web page from the local machine through a locally run web server.

[1] I use "markup" here to refer to both HTML and CSS together.

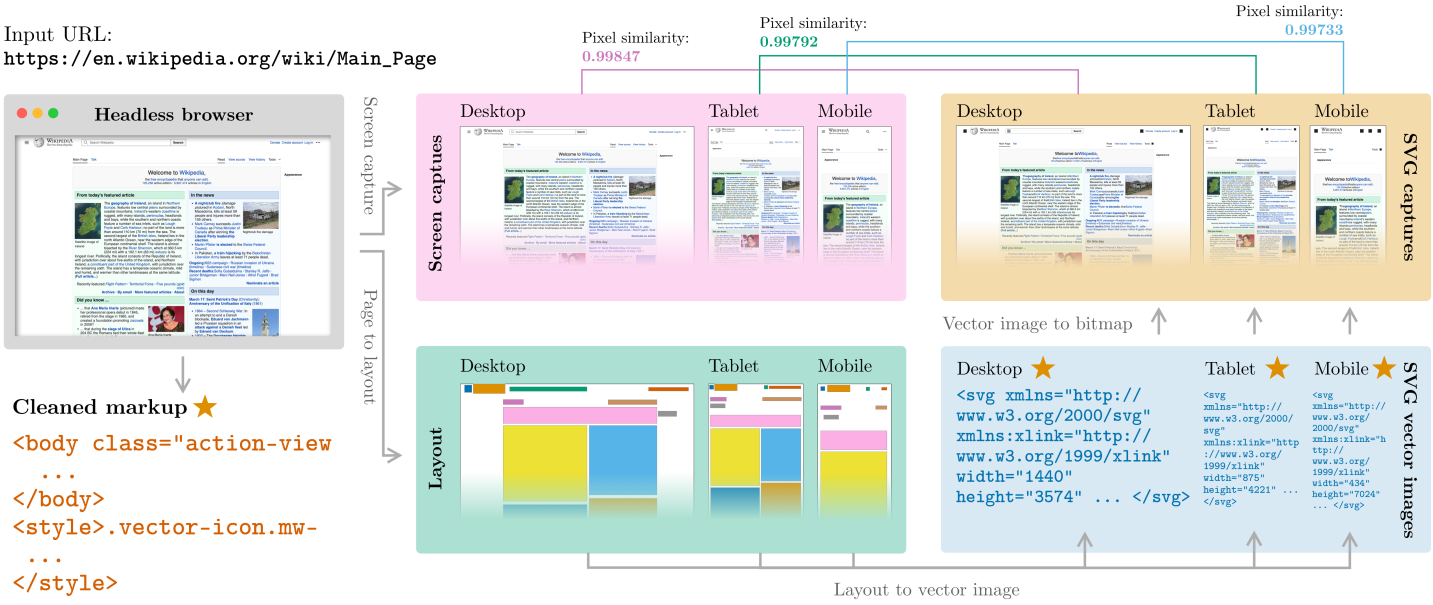


Figure 4.1: Conceptual figure of rendering a web page into pairs of vector images and markup. Wikipedia's homepage is used as an example. The star symbol indicates the principal data included in the data set. This example generates three different vector images for different viewport sizes.

8. Takes a **screen capture** of the entire web page, including parts outside the viewport, and saves it to the disk.
9. Converts the page layout to **SVG** element by element and saves the resulting vector image to the disk.
10. Opens the SVG file in the browser and takes a **screen capture of the image** as well.
11. Measures the **similarity** between the screen capture of the web page and the SVG representation. See Chapter 5. DR2
12. Measures whether the resulting screen captures **appear blank**. DR3
13. Saves the **metadata** to the disk.
14. Uploads metadata and the resulting HTML, SVG(s), and bitmaps to a cloud storage (Amazon Web Services Simple Storage Service or S3[2]).
15. May **repeat** from Step 6 with a different viewport size or an added data augmenter.
16. Once all viewport sizes and data augmenters are iterated on, it moves to Step 2 with the **next URL**.

The crawling and rendering tool I built uses a Chromium-based headless browser[3] called Puppeteer[4] and processes web pages concurrently. It is built in NodeJS and relies on a

[2]https://aws.amazon.com/s3/
[3]A web browser without a graphical user interface.
[4]https://pptr.dev/

web crawler framework called Crawlee[5] for queue management and browser management. Figure 4.1 explains the process visually.

Robots Exclusion Protocol Websites may host a `robots.txt` file in their document root that defines which automated software ("robots") are allowed to access their site and which pages. The protocol is oup-out and relies on voluntary compliance. Due to ethical aspects (see section A.2) of public data collection, I considered it essential to follow the protocol in a broader sense: I omitted pages that do not allow Google's and OpenAI's crawlers instead of just checking for definitions pertaining to my crawler. When crawling web pages, I clearly communicated the tool's name to the server through the user agent string. Robots Exclusion Protocol prevented the crawling of approximately 1.5% of the public URLs I attempted to render. DR9

Saving the web page Saving the web page involves several steps to make it compact, static, and self-contained. All external CSS files are downloaded and concatenated in a single `<style>` tag at the end of the HTML. URLs in the CSS files (e.g., images or imports) are updated relative to the page. All CSS rules that are not used in the given page are purged. The HTML markup is compressed by removing unnecessary white space as well as elements that might pollute the dataset, such as `<iframe>`, `<script>`, or `<video>`. Only the markup within the `<body>` tag is preserved.

Images embedded in the web page are downloaded to the disk with a short name from their hashed URL, and their references are updated in the markup. The image files are discarded after the web page is crawled and do not appear in the final dataset. This decision presented challenges when validating the model's results.

The saved SVG files are also cleaned up to take as little space as possible.

Blank pages When working with public URL lists (see section 4.4), I noticed that many pages appeared blank even when their markup was not empty. Since these pages do not contribute to the dataset, I filtered them out. I did this by checking if each pixel on their respective screenshots is of a single homogeneous color. DR3

Metadata Besides the resulting HTML and SVG files and their respective screen captures, the dataset contains the following metadata for each page:

- the original URL
- a unique ID based on the hashed original URL

[5] https://crawlee.dev/

- calculated similarity scores between the HTML and SVG screen captures for each viewport size
- the pixel dimensions of each screen capture for each viewport size
- whether the page was augmented with any of the data augmenters, and if so, which one
- whether any of the screenshots were blank
- the length of both the page and the SVG files in bytes

Crawling infrastructure My personal computer proved to be insufficient and impractical for crawling and rendering a large number of web pages within a reasonable amount of time. Due to high concurrency, memory became a bottleneck. Web rendering is a complex task, and the CPU quickly became another constraint, overwhelmed by computing layouts. Internet speed had a direct correlation to the speed of the crawling. For those reasons, I rented dedicated crawling servers with a much more powerful configuration: 13th generation Intel Core i5-13500 processor, 64 GB DDR4 RAM, 512GB Gen4 NVMe SSDs. The servers worked in parallel on different URL list partitions. DR8 Figure 4.2 explains the crawling process visually.

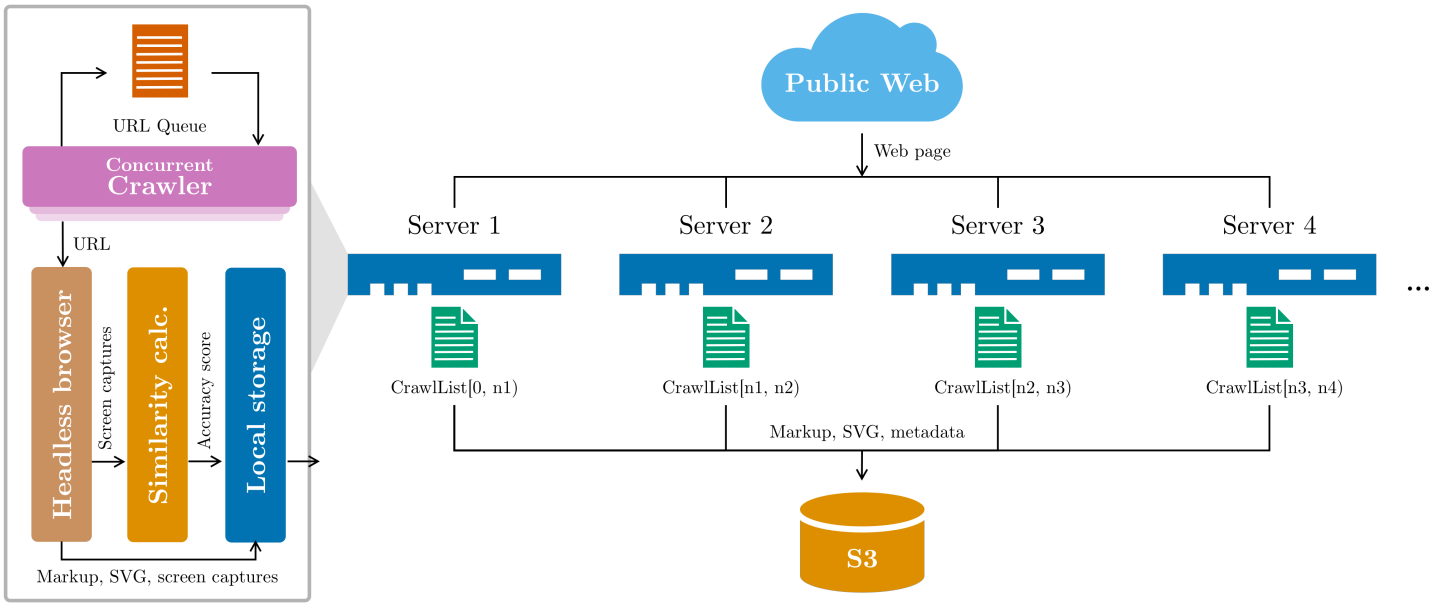


Figure 4.2: Conceptual parts of the crawling infrastructure

4.1.2 Data augmentation

Data augmentation can multiply the number of training data instances obtained from a single web page. DR5 One straightforward idea involves randomly altering a web page's

elements during rendering and saving it as a new item. While data availability was generally not an issue when crawling the public web, data augmentation provided a considerable speed-up. Augmenting an already loaded page is usually faster than crawling a new page.

I implemented the following simple data augmentation techniques.

`ChangeCssRulesDataAugmenter` Parses the CSS rules on a page and changes properties randomly. For the sake of simplicity, this augmenter only changes color and size values, but it can be extended with more complex logic.

`DeleteRandomCssRulesDataAugmenter` Deletes some CSS rules on the page with a pre-defined probability.

`DeleteRandomNodesDataAugmenter` Deletes some HTML elements on the page with a pre-defined probability.

`PermuteCssRulesDataAugmenter` Randomly permutes the order of all CSS rules on the page. CSS rules are order-dependent.

`PermuteNodesDataAugmenter` Randomly permutes the order of all elements on the page.

I assigned a weight to each augmenter and randomly selected one for each crawled web page, effectively doubling the size of the produced data for a given list of URLs. Different strategies might work here: all data augmenters can be executed for each page, creating a new data instance for each; a data augmenter can be executed multiple times, producing different random results; data augmenters can be combined.

The augmenters above are meant as proof-of-concept. Many other augmenters can be imagined, such as changing the website's text, inserting random elements, combining multiple web pages, etc.

Besides the simple augmenters above, I created two special classes to reduce the size of the produced data. In transformers, time and space complexity of the self-attention mechanism scale quadratically, $O(n^2 d)$ and $O(n^2)$ respectively where n is the input sequence length and d is the embedding dimension [26]. For resource limitations detailed in section 6.1, the goal was to reduce the input sequence size as much as possible. DR4

`MarkupSizeReducerDataAugmenter` This class aids in preparing datasets with a specific maximum markup length. It removes all HTML elements that are not in the current viewport. If the desired length is not reached, it also truncates all text nodes. If the resulting markup is still too long, it progressively deletes elements starting from the bottom of the markup until the limit is reached. The augmenter does not directly change the page's CSS, but since unused CSS rules are pruned when saving the page, the overall stylesheet length will be reduced, too. The corresponding SVG file is expected to decrease in size accordingly.

`PageChopper` This class is meant to generate datasets with items that are not just short but also simple. It recursively collects each visible element (and its inner elements) on a page, filters them against a size limit parameter, and then saves versions of the page where only one of these elements appears. The corresponding stylesheet is preserved for each element. This augmenter differs from the previous ones in that it may create a large number of input instances from each page.

4.1.3 Responsive web design

Responsive web design was briefly discussed in Chapter 2: user interface designers typically deliver different designs for different screen sizes. Through the use of CSS media queries [10, 7 Media types], identical markup code can be delivered to various device types, and the design can adapt to the available *screen real estate*.

The renderer described above can output different SVGs depending on the viewport size of the browser. DR7 An ideal model can use several SVGs as input and output the singular, responsive markup.

When creating datasets, I decided to capture vector images for the following viewport sizes for each page:

- 1440×900 pixels (WXGA+ Desktop)
- 834×1210 pixels (iPad Pro 11-inch M4)
- 393×852 pixels (iPhone 15)

Note that these are logical CSS pixel values as opposed to the physical pixel dimensions of the device [10, 4.3.2 Lengths].

The crawler was released under the Apache 2.0 license here: https://github.com/tcz/rb-crawler DR10

4.2 Ideal dataset size

Obtaining training data from the public web means there is no practical upper limit for training data size. Similarly, synthetic data generation makes data availability a non-issue. Nevertheless, it needs to be decided how much data is needed to train a performant model. DR5 There is no straightforward formula to determine the ideal data size for training a particular model but Large Language Model scaling laws have been studied in depth.

Hoffmann et al. [27] argue that many existing large language models are under-trained and suggests scaling the training data size proportionally to the model size. It finds that the

optimal pre-training data size in tokens is approximately 20 times the number of trainable parameters N. Muennighoff et al. [28] point out that a much smaller dataset can get comparable results, and the difference in loss between $20 * N$ and $5 * N$ is not significant.

Zhang et al. [29] point out that while the findings in Hoffmann et al. [27] hold for pre-training, when fine-tuning models, the size of the model has a larger impact on the performance than the dataset size. Intuitively, this rings true since a fine-tuned model relies on previously distilled knowledge in the base model and can take advantage of generalizations previously learned.

I planned to test several different model architectures, both pre-training and fine-tuning. The parameter count of these models ranges between 60 million and a couple of billion, although the larger models are used in fine-tuning, not pre-training. The number of trainable parameters is much smaller for these, in the range of tens of millions. See Chapter 6 for details on model selection.

Another data point I considered when determining the ideal data size was the size of other datasets for similar purposes. All models described in Related work (section 2.2) used bitmap inputs. Some use decidedly small training datasets, so I did not consider their data sizes instructive.

However, natural text machine translation can be a fitting analogy for translating vector files to HTML and CSS, since both have text modalities for input and output. It is important to note that the nature and complexity of human language translation differ significantly from approximating a browser rendering engine. Still, popular dataset sizes can indicate the order of magnitude of the data to be collected. Table 4.2 shows the token count of several popular public datasets for translation. There are several orders of magnitude of difference between the smaller and larger datasets that the machine learning community finds useful for translation.

Dataset name	Largest subset	Size (total tokens, millions)
Proyag/paracrawl_context [30]	eng-deu.both_contexts	4,666
Helsinki-NLP/opus-100 [31]	en-ur	62
Helsinki-NLP/news_commentary [32]	ar-ru	19
Helsinki-NLP/opus_books [32]	en-hu	10
dsfsi/vukuzenzele-sentence-aligned [33]	tsn-tso	0.4
google/smol [34]	smoldoc___en_xsr-Tibt	0.2

Table 4.2: The total token size of some popular translation datasets. I calculated the token counts by concatenating translation pairs and tokenizing each with Google's BERT multilingual base model tokenizer. Additional data in the dataset (e.g., context, metadata) was not considered.

In addition to scaling laws and machine translation datasets, I considered the project's time and financial constraints. More than anything else, this factor limited the size of the most extensive datasets.

Based on the reasoning above, I concluded that the ideal dataset needed to contain at least several hundred million to several billion tokens.

4.3 Synthetic data

Instead of crawling large numbers of real-world websites, a simpler and more controlled approach is to generate web pages from scratch. This approach let me manage complexity and focus on a smaller set of web features. DG1

4.3.1 Simple synthetic data

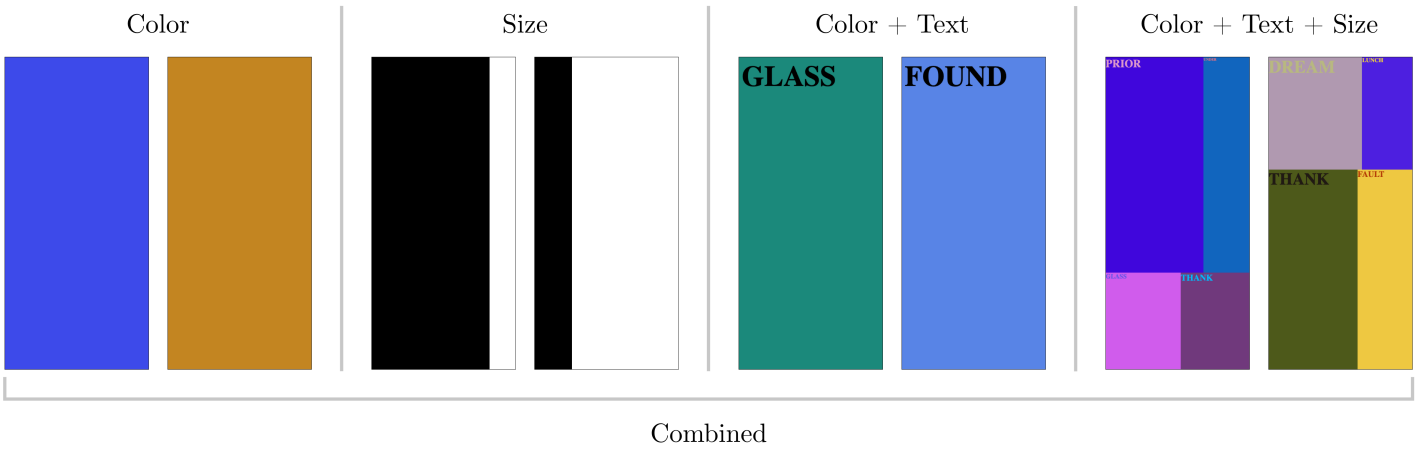


Figure 4.3: Examples of web pages included in each simple synthetic data set.

To prove that transformers can learn to translate between simple vector images and their corresponding web pages, I created a set of basic training datasets with increasing complexity. The web pages were generated using templates with placeholders for randomly chosen variables. Soselia, Saifullah and Zhou [19] use a similar approach. The resulting pages were then run through the renderer as described in section 4.1 to produce their SVG images. For these datasets, data augmentation was not used.

The process resulted in a dataset with very similar items with large repeating sections between instances (headers, basic structure, etc.), and only minor differences between each instance. This is very distinct from the diverse input dataset with the full range of web features we expect a fully functional model to handle. Nevertheless, these datasets helped me test the hypothesis in its simplest form and discover the limitations of the model training.

The simplest possible web page I could devise had no content, only a single CSS variable setting the page's background color. The color is randomly chosen from 2^{32} possible colors. This dataset was used to prove that transformers can learn to match corresponding sections between markup and HTML and to translate between different color representations (hexadecimal and red-green-blue notation).

I also created a special responsive version of this dataset in which the page appears with a different, randomly chosen background color on mobile and desktop. This was meant to test multiple vector inputs and a single markup output with responsive features.

Calculating the layout of a web page involves simple arithmetic: additions, multiplications, divisions, etc. Consider an element styled as `50vw` of width and `100pt` of height. The width represents 50% of the viewport width. To get a size in pixels to be painted, the layout engine performs a multiplication: $0.5 * V_{width}$ where V_{width} is the width of the browser's viewport in pixels. Similarly, the height in pixels will be calculated like so: $100/R_{points\ to\ inches} * R_{pixels\ to\ inches}$ where $R_{points\ to\ inches}$ and $R_{pixels\ to\ inches}$ are constants defined in the CSS specification [10, 4.3.2 Lengths]. More complex calculations determine the flow of the elements on the page, align the text, etc. Getting from a layout (vector image) to a markup (HTML and CSS) requires the same calculations in reverse. The bottom line is that our models need to be able to perform arithmetic operations, or approximate those.

Therefore, the following simple dataset I created was meant to test if the model can learn elementary arithmetic when it takes an SVG input and generates markup that uses length units other than pixels.

Aligning, typesetting, and displaying text are important features of a browser engine, so the next dataset extends the color dataset by displaying a random word.

In the most complex synthetic dataset, color, text, and size definitions are combined. Pages display four colorful sections with different sizes set using distinct units and random text in each section with their own random colors and sizes.

Learning to translate inputs with a homogeneous pre-defined format is much simpler than learning to handle arbitrary inputs. To test whether the model can learn from a more heterogeneous dataset, I created one that combines all the above datasets.

Figure 4.3 shows examples of the synthetic datasets rendered in a mobile viewport.

The simple templates I used to generate the synthetic pages may be further extended with control structures such as `IF` conditionals and `FOR` loops. Existing templating engines such as Jinja [35] may be used. Still, template-generated pages will always result in a relatively low complexity dataset compared to a collection of real-world websites from the public web. Training *low-perplexity* [36] models on a simple synthetic dataset may be quick and inexpensive, but the model's usefulness will be minimal. These datasets were meant to derisk the step of moving to larger, more complex datasets and larger models.

4.3.2 Other methods

Beyond simple templates, there may be other methods for generating more complex synthetic data.

Probabilistic Context-Free Grammar One such method is probabilistic context-free grammar [37] or PCFG, an extension of context-free grammar where production rules are assigned a probability and executed stochastically.

Here is a simple example of a simple PCFG to produce a web page.

```
Body -> "<body>" Compound "</body>" [1.0]
Compound -> Content Content [0.7] | Content [0.3]
Content -> Div [0.5] | "text" [0.5]
Div -> "<div>" Compound "</div>" [1.0]
```

An example product of the above grammar looks like this (formatted for clarity):

```
<body>
<div>
    <div>
        <div>
            <div> text
                <div> text
                    <div>
                        <div> text
                            <div> text text</div>
                        </div>
                        <div> text</div>
                    </div>
                </div>
            </div>
            <div>
                <div> text text</div>
                text
            </div>
        </div>
        text
    </div>
    text
</div>
text
</body>
```

Constructing a set of complex production rules by hand can be difficult and time-consuming. In terms of complexity and variability, the resulting synthetic data will likely

fall behind a data set of real-world websites.

Instead of constructing rules by hand, they can be learned from an existing corpus, such as a previously crawled set of websites. Both the set of rules and the probabilities can be learned without having to manually construct them [38]. This requires a previously obtained corpus of web pages.

Due to the constraints of this project, I did not further investigate probabilistic context-free grammars nor generate a dataset using PCFG.

Transformers A generative model based on the transformer architecture might be a reasonable choice for constructing abundant, synthetic training data. A pre-trained, decoder-only transformer with a high-temperature setting can stochastically generate new web page instances.

The transformer itself does not ensure the syntactical correctness of these pages, but the resulting code may be checked for correctness and either discarded or corrected with static analysis tools. Training such a model requires a large, previously obtained corpus.

To test this concept, I fine-tuned a model with a dataset obtained from crawling public websites (see section 4.4). Fine-tuning is a supervised training technique where a previously pre-trained base model undergoes additional training for a specific task. For the base model, I chose Codestral 22B v0.1 [39] and fine-tuned it using Low-Rank Adaptation (LoRA). For more on LoRA fine-tuning, see section 6.2.2. Codestral is a decoder-only, instruction-tuned [40] transformer model with open weights. It was trained on 80+ different programming languages with the stated purpose of aiding programming tasks, such as code completion, explaining, documenting, etc. I chose this model, assuming it has learned syntactic patterns and valid generalizations over program code.

The training dataset contained 25,000 web pages saved using the tool described in section 4.1. The instruction prompt for the training and generation was:

```
[INST]Your job is to generate random, valid HTML+CSS code for data set
    generation. Ensure the HTML and CSS have valid syntax and are compatible
    with modern web browsers. Only generate the <body> tag and its contents, the
     <html> and <head> tags are not needed. The CSS should the very last element
     in code generated and should be enclosed in <style> tags.

You must generate only HTML and CSS code within <answer> XML tags.

Generate HTML and CSS.[/INST]
```

The fine-tuning was done on a single Nvidia H100 GPU for one epoch and took 21 hours[6].

It may be noted that fine-tuning was not strictly necessary. The base model is capable of generating markup with the above prompt. However, when testing the base model, it

[6]The fine-tuning code was published in the training experiment repository under https://github.com/tcz/rb-experiments/tree/main/20250313-codestral-fine-tuning

	Hallucinations (%)	Self-similarity (avg. F1-score)
Base model		
temperature=0.1	85%	0.8247
temperature=0.4	61%	0.8395
emperature=0.8	77%	0.8290
Fine-tuned		
temperature=0.1	0%	0.9902
temperature=0.4	6%	0.9279
temperature=0.8	32%	0.8835
Sample of web pages	N/A	0.8045

Table 4.3: Output quality of the base Codestral model and the fine-tuned model with different temperature settings. Self-similarity was calculated using BERTScore. Higher score means more self-similarity. Hallucinations were quantified manually as a percentage of the sample. A random 100-item sample of crawled public web pages was also measured for self-similarity for comparison.

often emitted random additional instructions before the frontend code or code in languages other than HTML and CSS. This is a common LLM issue referred to as hallucination [41].

I measured the extent of hallucinations on a sample of 100 model outputs with three different temperature[7] settings (0.1, 0.4, and 0.8) using subjective classification. I also measured self-similarity to confirm whether the model creates diverse enough data compared to data obtained from the public web. For this, I used a contextual embedding score called BERTScore [42]. I calculated the average F1 score between each output pair, only considering the contents of the `<answer>` tags. I did the same on the fine-tuned model. Results are summarized in Table 4.3.

In summary, the fine-tuned model had a lower rate of hallucinations, although not zero, except at the lowest temperature setting. Unfortunately, at this temperature, responses became repetitive and useless for training data generation. The output became short and simple, referencing "Not found" and "Access denied" pages, which may have been overrepresented in the training input. In conclusion, I found that neither the fine-tuned model nor the base model was useful for generating synthetic data. They were hindered by hallucinations, self-similarity, and relatively short and simplistic outputs.

A different base model, longer training time, longer sequences, better-filtered input data, and better-written prompts might result in a usable model generating synthetic training data of web pages.

Still, the principal reason I discarded transformers for synthetic data generation was their cost. Generating a single instance took approximately 8 seconds on a single Nvidia H100. Crawling can be much faster when parallelized and run on much cheaper hardware.

[7] An adjustable parameter controlling the level of stochasticity of the generation. Chapter 7 contains a short explanation on what *temperature* means in this context.

Using transformers for synthetic data generation remains an interesting option, assuming inference costs will decrease in the future, or if the legal or ethical environment changes, so that it no longer favors crawling public data. I discuss this subject in Appendix A.3.

Manual contruction While not a scalable approach, synthetic data can be created manually. When a small dataset size suffices and control over the complexity and features is essential, this can be a reasonable choice. This was precisely the case when I was constructing the validation split for the public web dataset.

The validation split of a dataset is used to evaluate the model's performance during training, to tune hyperparameters, and to prevent overfitting. Ordinarily, it is a smaller (e.g., 10%), randomly selected partition of a larger dataset that gets excluded from the training data.

When training models on the public web datasets, I opted to construct a small validation dataset manually. I list the reasons below.

1. Calculating accuracy metrics (see Chapter 5) during the training is a costly and slow process involving generating predictions, rendering them in a browser engine, and then processing the resulting screenshots. A large validation set with long, complex examples would have unduly slowed training.
2. When training on various public web data sets, I needed a standard validation set across all of them to assess the performance of the model. Using samples from the datasets themselves would have made it impossible to compare experiments that were trained on different datasets.
3. Ultimately, the measure of success was that the models learn to construct relatively simple webpages from a vector image input. Using a random set of, potentially very complex and large, public webpages for validation would have made it difficult to assess if the model is converging towards that goal.
4. Due to how the datasets were constructed, external resources, notably images, had to be discarded. This meant that rendering random webpages referencing missing resources would have caused rendering errors, affecting accuracy metrics[8]. For a small set of manually constructed validation web pages, external resources could be locally stored and served during validation.

For those reasons, I constructed a small validation dataset of simple websites. Figure 4.4 shows them in desktop resolution. I will refer to this dataset as "*Small Validation*".

[8]Note that this is not a concern when the final model is used in a design workflow. In that situation, the images referenced in the vector image can be preserved and included in the HTML output. I demonstrate this in Chapter 7.

Figure 4.4: Web pages in the "Small Validation" dataset.

4.3.3 Synthetic datasets

Synthetic datasets were published under the Apache 2.0 License at the following URLs.

DR10

Color	https://huggingface.co/datasets/tcz/rb-color
Color responsive	https://huggingface.co/datasets/tcz/rb-color-responsive
Size	https://huggingface.co/datasets/tcz/rb-size
Color+Text	https://huggingface.co/datasets/tcz/rb-color-text
Color+Text+Size	https://huggingface.co/datasets/tcz/rb-color-text-size
Small Validation	https://huggingface.co/datasets/tcz/rb-small-validation

Table 4.4 shows the sizes of the synthetic datasets.

	Items	Total Tokens (millions)	Median Tokens Per Item
Color	100,000	35	351
Color responsive	100,512	71	709
Size	99,789	43	431
Color+Text	100,000	67	670
Color+Text+Size	99,853	217	2,178
Small Validation	4	≈ 0	1,406

Table 4.4: Total token sizes and median item token lengths of the synthetic datasets. Google's BERT multilingual base model tokenizer was used to calculate token counts. For the larger datasets, a sample of 5,000 random items was used to calculate token lengths.

4.4 Public data

Crawling the public web for diverse training data is a logical choice with some ethical and legal caveats (see section A.2). The public web is a nearly inexhaustible source, containing many more potential training data instances than I would need (see section 4.2 for ideal dataset size).

I obtained the list of URLs to crawl from Common Crawl [43]. Common Crawl is a non-profit that maintains large databases of crawled websites. I only needed the URLs,

not the crawled web content, since I had to render each website, a task that Common Crawl is not performing. I used the August 2024 Crawl Archive (CC-MAIN-2024-33), which contained 2.3 billion web pages.

To get the URLs to crawl, I downloaded several arbitrarily chosen index files from Common Crawl with `.com` prefix. I iterated over the items in these index files and:

- Skipped items where the original status code is other than `200 OK` [44, 6.1.1 Status Code and Reason Phrase]
- Skipped URLs ending in `robots.txt`
- Skipped URLs from any previously seen domain names. The rationale was that web pages hosted under different domain names are more likely to have a different design than pages under the same domain, which is preferable for the variety of the dataset.
- If the URL was not skipped, I added it to the crawl list. Note that the URL was not guaranteed to be a domain root, e.g. `http://www.example.com/`. Depending on the ordering in the index file, non-root pages like `http://www.example.com/page.html` were common.

The resulting URL list was crawled and rendered using the methods described in section 4.1.1. Several smaller and bigger datasets were created. Crawling the largest dataset with 615,945 items took approximately one week using the hardware described above.

A mistake I noticed after the crawling was that, even though non-200 responses were filtered from the original list, they were not filtered during crawling. If a server was reachable but returned, e.g., a `404 Not Found` status, the page ended up in the data set. `404 Not Found` pages often contain design elements, so I did not consider this a serious issue. They enrich the dataset like any other page, although they may introduce repetition in layout and text content. If I did a new crawling batch, I would opt for excluding non-200 responses.

Unlike synthetic pages, public web pages may contain embedded elements. As described in section 4.1.1, most embedded elements such as scripts, videos, `iframes`, and external fonts were removed. Images are an essential design element, and their intrinsic size may influence the page's layout. For those reasons, embedded images were kept and downloaded to temporary offline storage during the rendering. However, they were excluded from the final dataset for practical file size limitations.

Once the crawling was finished, the crawled dataset was then filtered to exclude blank screenshots, representing 16.2%. DR3 Next, items with low similarity were filtered. DR2 These were web pages where the markup to SVG conversion performed poorly, resulting in a significant difference between the page's screen capture and the SVG's screen capture. The similarity cutoff point of 0.99 *Multi-Scale Pixel Similarity* (MSPS) was determined by sampling a large number of screen captures and using subjective judgment. MG2 See

Chapter 5 for more information on the similarity metrics. All three crawled viewport sizes (mobile, tablet, desktop) were filtered by MSPS. This excluded 34% of the original dataset, resulting in a final dataset of 353,908 items. Figure 4.5 shows the distribution of MSPS scores.

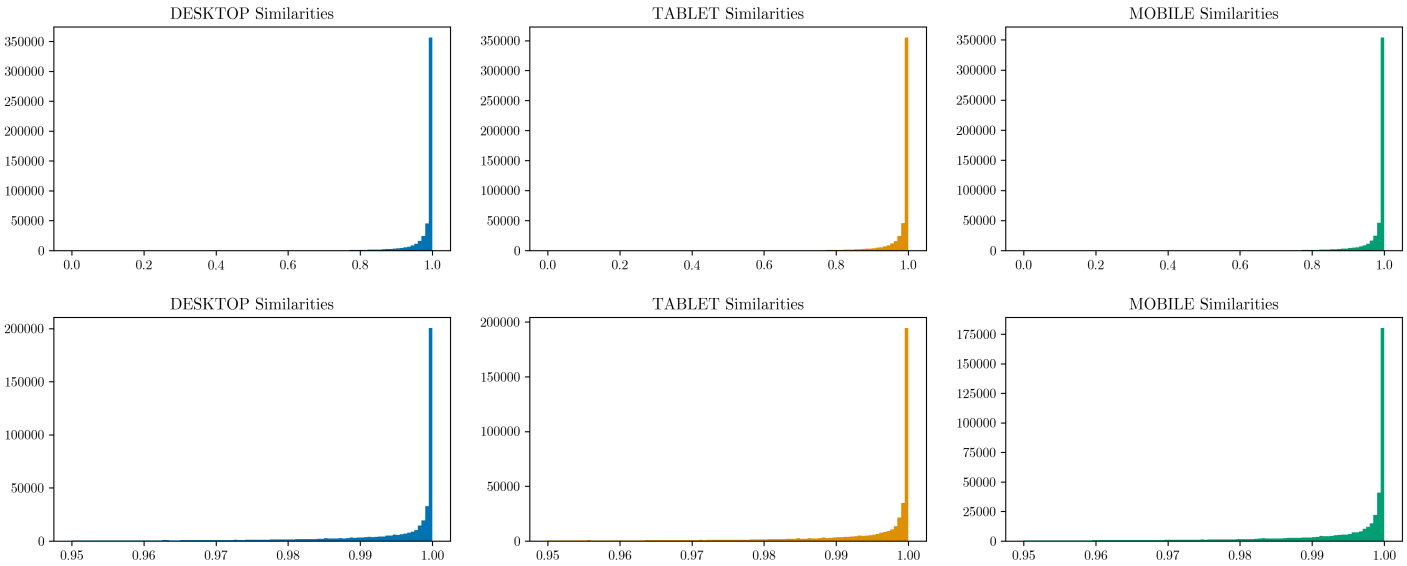


Figure 4.5: Multi-Scale Pixel Similarity scores of the *large* dataset after filtering out pages with blank screenshots.

Besides the "Large" dataset described above, I created two separate datasets using the size-limiting augmenter classes: `MarkupSizeReducerDataAugmenter` and `PageChopper`. These were parameterized to limit markup byte lengths to 8,000 and 1,000, respectively. These datasets were based on new sections of the URL lists and followed the same procedure as the "Large" dataset. Crawling took approximately one week for each. DR4

4.4.1 Public web datasets

Public web datasets were published under the Apache 2.0 License at the following URLs. DR10

Large	https://huggingface.co/datasets/tcz/rb-large
Reduced	https://huggingface.co/datasets/tcz/rb-large-reduced
Chopped	https://huggingface.co/datasets/tcz/rb-large-chopped

Table 4.5 shows some size statistics.

	Items	Total Tokens (millions)	Median Tokens Per Item
Large	353,908	101,874	150,868
Reduced	392,112	20,167	47,821
Chopped	214,812	4,293	11,916

Table 4.5: Total token sizes and median item token lengths of the public web datasets. Google's BERT multilingual base model tokenizer was used to calculate token counts. All SVG resolutions have been included in the items. Statistics are calculated using a sample of 5,000 random items from each dataset.

5 Metrics

5.1 Accuracy metrics

Accuracy metrics measure visual differences between the input vector image and the output webpage. Designers may expect a *pixel-perfect* representation of their designs in the final web product. This means that there is no discernible difference between the design's bitmap representation and the web page's bitmap representation down to the level of individual pixels.

A reliable accuracy metric was required for several different purposes: MG1

1. To assess the quality of markup to SVG conversion when generating training data pairs as seen in Chapter 4
2. To track the model's evolution during training
3. To evaluate the performance of a trained model
4. To improve final performance with best-of-N ranking [45]

A naive approach could compare the bitmaps pixel by pixel and calculate the mean squared error (MSE) like so:

$$\text{MSE} = \frac{1}{H \cdot W \cdot C} \sum_{h=1}^{H} \sum_{w=1}^{W} \sum_{c=1}^{C} \left(I_1(h,w,c) - I_2(h,w,c)\right)^2 \tag{5.1.0.1}$$

Here H, W, and C respectively refer to the pixel height, pixel width, and the number of channels of the compared images, while I_1 and I_2 refer to the pixel at location (h, w), channel c for the two images.

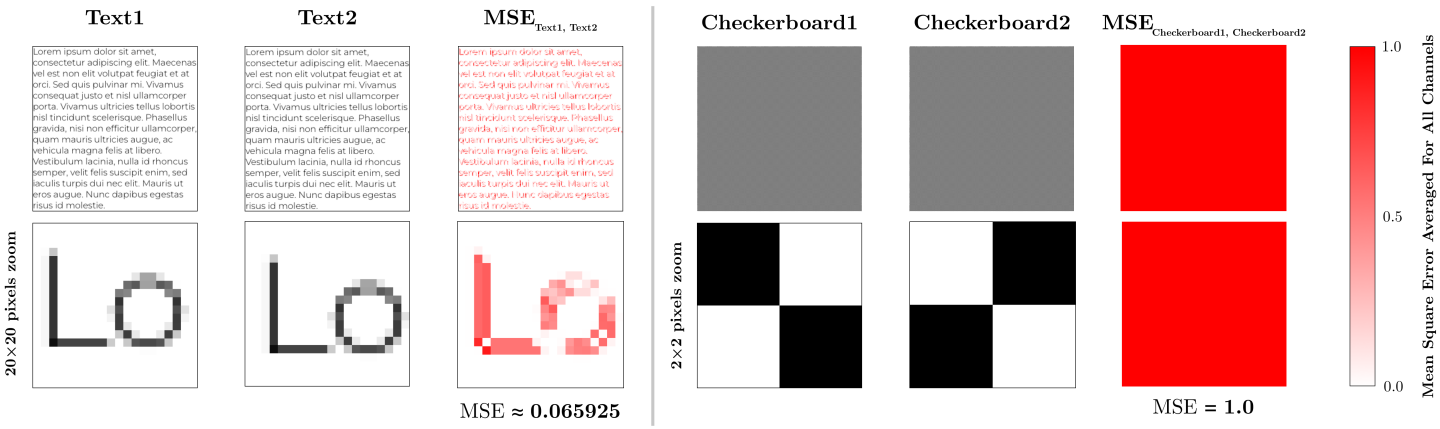


Figure 5.1: Visualization of mean squared error with two example image pairs. In the first example, Text1 and *Text2*, the images are identical, except that the text on *Text2* was translated one pixel to the right and down compared to *Text1*. This is a negligible difference to a human observer and would indicate excellent model performance, but the calculated MSE is relatively high in this example. The second example takes the problem to the extreme. The two images show a checkerboard pattern with pixels alternating between black and white in each row. *Checkerboard1* starts with black in the top left corner, while *Checkerboard2* starts with white. The resulting images look nearly identical to a human observer, yet the calculated MSE is 1.0, the highest possible value.

This approach may result in a substantial loss (a difference between prediction and target images) even for minor differences that are practically invisible to a human observer. Figure 5.1 illustrates this problem. While our ultimate goal is a truly *pixel-perfect* conversion from design to markup, this may not be an immediately realistic goal. An ideal metric measures the evolution of the model gradually and measures reasonably low loss when the model is not yet perfect, but emits an output similar but not identical to the target. MR5

5.1.1 Perceptual similarity

Many complex algorithms are available that compare a reference image to a potentially degraded version. These algorithms try to estimate the subjective judgment of a human observer. They are called Full-Reference Image Quality Assessment (FR-IQA), where "Full-Reference" means we can access the original reference image. In our case, the reference image may be the original design as delivered by the designer. It may also be the screen capture of the original web page during dataset construction, which is used to measure the quality of the conversion from markup to SVG.

Not all FR-IQA algorithms are a good fit for our use case. Some of these algorithms are designed to be robust against rotations and translations. This is undesirable for our use case since rotations, scaling, and large translations between a design and the resulting markup should be considered a defect. An ideal FR-IQA algorithm produces

high loss with rotations, scaling, and large translations, but it is robust against small, few-pixel translations. MR3 MR4

 Reference	 Rotated	 Translated	 Squeezed	 Scaled	 1px-trans.	Time (s)
TOPIQ [46]	0.7438108	0.7448106	0.7420042	0.7495018	0.1015147	0.247
AHIQ [47]	0.9514484	1.0921408	1.0229274	0.9072852	0.5039899	0.269
PieAPP [48]	0.7264122	0.7321822	0.5625778	0.8485057	0.060036	0.146
LPIPS [49]	0.6214625	0.7138035	0.6818361	0.8698413	0.0407809	0.108
DISTS [50]	0.1754472	0.1252416	0.254124	0.2622414	0.0069962	0.110
WaDIQaM [51]	0.2324741	0.2426802	0.2575099	0.2473186	0.1206883	0.140
CKDN [52]	0.6892447	0.6187835	0.6525854	0.6210397	0.3293804	0.121
FSIM [53]	0.4262481	0.484176	0.4085104	0.5063123	0.0601805	0.248
MS-SSIM [54]	0.6318116	0.7013864	0.6851234	0.7607858	0.0339208	0.249
CW-SSIM [55]	0.7225356	0.7532272	0.7921229	0.6809691	0.0026404	1.567
VIF [56]	0.9844843	0.9837387	0.9828228	0.9838052	0.7154267	0.282
GMSD [57]	0.3110082	0.3265839	0.3245934	0.3145258	0.1130009	0.102
NLPD [58]	0.9626138	0.9904734	0.9079056	1.091566	0.2708094	0.182
VSI [59]	0.2047782	0.2268612	0.2288077	0.2067673	0.0134498	0.201

Table 5.1: Comparing several popular FR-IQA algorithms. A reference image was transformed in various ways, and the loss between the reference and the transformed image was calculated. *1px-trans.* means that the reference image was translated by one pixel to the right and one pixel to the bottom. All loss scores were normalized to 0.0-1.0 where possible and standardized to mean "less is more similar." The reference image was taken from the Weber [60] dataset. The time of a single comparison was calculated on a machine with an Nvidia H100 GPU.

Table 5.1 compares popular FR-IQA algorithms in terms of their behavior across image transformations. AHIQ and CKDN were discarded because they produce variable non-zero loss when the reference image is tested against itself. DISTS, WaDIQaM, GMSD, and VSI were discarded because they are highly robust against large transformations like rotations and scaling. VIF and NLPD were discarded because of their relatively high loss of the 1-pixel translated image. Among the remaining candidates, LPIPS was chosen based on its time performance.

LPIPS [49] (Learned Perceptual Image Patch Similarity) takes advantage of deep features from convolutional neural networks (CNNs) trained for image classification. Input images first pass through the convolutional layers of the image classification model (in our case, AlexNet), resulting in feature embeddings for both the reference and test images. Then, the distances between these embeddings are calculated. These distances are then input to a model trained to map embedding distances to human perceptual similarity scores. MR6

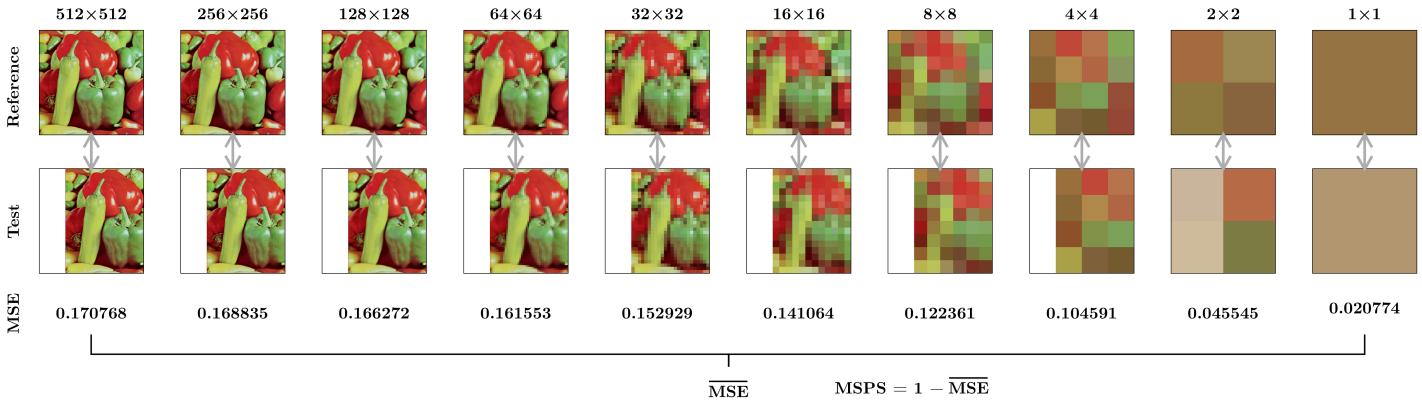


Figure 5.2: Visualization of Multi-Scale Pixel Similarity (MSPS).

5.1.2 Multi-Scale Pixel Similarity

LPIPS is appropriate for tracking training loss, evaluating model performance, and n-best reranking. However, a stricter metric was needed to assess conversion from markup to SVG during dataset generation. FR-IQA algorithms are typically developed to evaluate image compression or restoration, and they consider the idiosyncrasies of the human visual perception system. Signals such as pixel values are weighed to mimic the behavior of the human eye and minor visual differences might be overlooked.

Using FR-IQA algorithms to evaluate markup-to-SVG conversion can pollute the training dataset with imperfect SVG-markup pairs. This could sabotage our goal of training a model with *pixel-perfect* image-to-markup conversion. Given the relatively low cost of crawling and data abundance, I chose to discard dataset items based on a stricter similarity score.

I opted to create a new metric that is based on the pixel-level comparison discussed in the introduction, but it solves the issue with the naive pixel-level MSE, and it is stricter than FR-IQA metrics. I call this metric *Multi-Scale Pixel Similarity* or MSPS. MR3

MSPS uses MSE but progressively reduces the sampling density of the input image. This is analogous to human squinting and is not a novel idea: Wang, Simoncelli and Bovik [54] take a similar approach and apply it to SSIM [61]. First, the pixel-level MSE calculate as described earlier. Then the images get repeatedly resized to half their horizontal and vertical dimensions, resampled and compared again. This repeats until the image measures one pixel in width or height. Finally, the mean of all the previously calculated MSEs is computed. This is a greater-is-better metric, so the average error is subtracted from one to give similarity. See Figure 5.2 for a visual explanation and equation 5.1.2.1 for the formula.

$$\mathrm{MSPS}\Big(I_1, I_2\Big) \;=\; 1 - \frac{1}{N}\sum_{i=1}^{N}\left[\frac{1}{H_i W_i C}\sum_{h=1}^{H_i}\sum_{w=1}^{W_i}\sum_{c=1}^{C}\Big(I_1^{(i)}(h,w,c) - I_2^{(i)}(h,w,c)\Big)^2\right] \quad (5.1.2.1)$$

Here I_1 and I_2 are the input images to compare, and $I_1^{(i)}$ and $I_2^{(i)}$ are their scaled down versions. $I_1^{(1)}$ and $I_2^{(1)}$ refer to the original images without scaling, while $I_1^{(2)}$ and $I_2^{(2)}$ are half the size of the originals, etc. Resizing is done by 2D average-pooling. Other symbols were explained in equation 5.1.0.1.

N is the total number of MSE calculations and can be calculated like so:

$$N \;=\; 1 \;+\; \left\lfloor \log_2\Big(\min(H_1, W_1)\Big)\right\rfloor \quad (5.1.2.2)$$

Here, H_1 and W_1 refer to the pixel dimensions of the original image.

Table 5.2 compares the scores from naive MSE to pixel similarity using the images from Figure 5.1.

	Text score	*Checkerboard* score
1 − MSE	0.934075	0.0
MSPS	0.989907	0.888889

Table 5.2: Comparing naive MSE with Multi-Scale Pixel Similarity (MSPS) for images from Figure 5.1

I released an implementation of MSPS under the Apache 2.0 License at https://github.com/tcz/rb-msps

5.1.3 Related works

Works described in section 2.2 use various metrics to evaluate the performance of their respective models.

Soselia, Saifullah and Zhou [19] employ a code-based metric and two image-based metrics. The code-based metric compares the expected and generated markups and calculates a quality score. This paper coins *htmlBLEU*, a syntax-aware version of BLEU-score [62]. They find that *htmlBLEU* correlates with the image-similarity metric to some extent, although they do not calculate the significance using a null hypothesis. This is not surprising, given their simple and standardized markup in the training data. It is doubtful that such code-based metrics remain useful for more capable models trained on real-world training samples. There are many ways to implement a given design in HTML and CSS.

This makes code-based metrics prone to Type II errors. The paper acknowledges the limitations of their approach. MR1

As for image-based metrics, Soselia, Saifullah and Zhou [19] use MSE as described above. They also introduce a metric based on the intersection between non-empty pixels in the reference and the tested image. They use the latter as input for model fine-tuning. Once again, such a metric might be useful for the specific, limited training data in the paper, but less so with real-world examples.

Robinson [16] uses MSE and SSIM [61] to evaluate their model performance. We discussed both metrics above. Gui et al. [21] uses SSIM and CLIP[63]. The latter uses the closing similarity of latent vectors of a multimodal (text and image) model to compare images. It does not list the reasons for choosing this metric, which seems better suited to measure the similarity of a text input against an image input.

A relatively recent paper by Roberts et al. [1] discusses the problem of benchmarking vision-language models on image-to-structure tasks. These tasks include generating web pages from images. The authors benchmark a bitmaps-to-code task among others. They introduce a new dataset called **Image2Struct** and develop two new metrics: cosine similarity between the inception vectors (CIS) and earth mover's similarity (EMS). CIS is similar to LPIPS in that it takes advantage of hidden layers of pre-trained image classifiers, but it's simpler than LPIPS because it only uses a single layer.

EMS is a more innovative metric: it is based on Earth Mover Distance [64] (EMD), which, in its original form, discards spatial pixel information and compares pixel value probability distributions in two images. EMS introduced by Roberts et al. [1] extends EMD with spatial information by subdividing the images into patches and incorporating their spatial distances into the EMD cost. This causes the metric to perform well when parts of the reference image are translated by small distances on the test image. Notably, the metric discards color information by converting images to greyscale first. Another issue with the metric is that it is very susceptible to the patch size chosen, as seen by the results of an experiment I performed in Table 5.3. Additionally, their implementation was measured to be notably slower[1] than MSPS or LPIPS. MR8

In addition to the two new metrics, the Image2Struct benchmark takes advantage of existing metrics like pixel similarity, SSIM, and LPIPS. It uses a different base neural net (VGG) and normalizes LPIPS to a scale where a greater value means more similarity.

[1]Mean wall time to compare a pair of 1920×1080 pixel images: 5.31 seconds in EMS, 0.45 seconds in LPIPS, 0.13 seconds in MSPS

Patch size	*Text* score	*Checkerboard* score
(5, 5)	0.912573	0.624331
(6, 6)	0.942947	0.713801
(7, 7)	0.913324	0.477627
(8, 8)	0.888891	0.540136
(9, 9)	0.888672	0.780074
(10, 10)	0.882436	0.830694

Table 5.3: Testing EMS for images from Figure 5.1 using different patch sizes

5.2 Other metrics

During the model training and inference, I paid attention to the following additional metrics:

- **Cross-entropy loss** It measures the difference between the target labels' predicted probability distribution and true probability distribution. The evolution of cross-entropy loss on the training and validation datasets can give important clues on whether the model is converging or overfitting.
- **Total training time and FLOPs** The time taken to train a model and the total FLOPs were closely monitored for cost management and to decide if reducing the input length or dataset size was necessary.
- **Inference time** Ingressed and generated tokens per second, and the time to generate a single design were measured to assess the practical value of the trained model.
- **Visual inspection** Besides the accuracy metrics selected above, I visually inspected the generated designs and the corresponding HTML+CSS code and made a subjective assessment of their quality.

6 Models and training

6.1 Model selection

With training data and metrics defined, the next step was choosing a model architecture to train.

A neural machine learning model's architecture describes its building blocks, such as layers of different types, the activation function used, and the connectivity pattern between them. The architecture determines a model's number of trainable parameters and learning capacity. Different architectures come with different sets of trade-offs: performance, resource requirements, scalability, and generalization capabilities are directly determined by a model's architecture.

Below, I describe the criteria I considered during the model selection. TG1

Existing or new Converting vector images to markup can be formulated as a sequence-to-sequence (seq2seq) task: matching an input sequence of vector image bytes to an output sequence of HTML and CSS code. There is a variety of open-source seq2seq model architectures to choose from. The most modern and powerful architectures are based on transformers [6].

The task becomes more complex when dealing with various screen sizes for responsive design (see section 4.1). TR2 For instance, when a desktop and a mobile design are available, we expect a single sequence of responsive markup to be produced from two distinct input sequences. In other words, the task becomes sequence**s**-to-sequence (plural-to-singular), mapping several input sequences to a single output sequence. No available, open models could handle multiple input sequences. It was uncertain whether the task could be restated

as sequence-to-sequence (singular). Building a new architecture or extending an existing one to support multiple input streams had to be considered.

A few related papers tackle this issue in specific problem domains: Chen, Fan and Panda [65] fuse two encoder branches with high and low image resolutions using cross-attention in order to improve image classification. Dens et al. [66] use a similar technique to improve peptide binding predictions. Hammad, Moretti and Nojiri [67] use fused encoders in the domain of particle physics. These techniques are typically limited to two encoder branches.

Ultimately, I performed experiments using simple synthetic data and found that sequence concatenation with segment embeddings produces acceptable results with multiple input image resolutions. Thus, I discarded creating a new sequences-to-sequence architecture and stuck with available seq2seq models.

Encoder-decoder or decoder-only The original transformer paper [6] describes an architecture with two branches: an encoder that extracts features from the input sequence and a decoder that produces an output, connected by a cross-attention mechanism. This architecture works well with sequence-to-sequence tasks such as translation. Nevertheless, other architectures can be built using only an encoder or a decoder branch. The former is typical in classification and sentiment analysis tasks. The latter has gained recent prominence in autoregressive tasks such as text generation, chatbots, and code completion. These applications found such commercial success[1] that current research overwhelmingly focuses on decoder-only models. Nearly all available large pre-trained language models are built in this architecture. State-of-the-art decoder-only models have surpassed encoder-decoder models in traditional sequence-to-sequence tasks like translation [68].

Since producing markup from a vector image can be formulated as a translation task, the first intuitive choice is the encoder-decoder. Nevertheless, I considered both architectures when choosing the models to train.

Pre-training or fine-tuning Pre-training starts with a model initialized with random trainable parameters and trains the model from scratch on the dataset. Fine-tuning is a supervised transfer-learning technique that takes a model previously pre-trained on a large dataset. It applies additional training using a distinct, domain-specific, labeled dataset, adjusting some or all model weights. Several large pre-trained models with permissive licenses are available for fine-tuning. They were trained on massive text (and sometimes multimedia) data from the public internet and other sources.

My initial intuition was that since the project takes place in a particular domain, general knowledge instilled in large pre-trained language models will not meaningfully contribute

[1] The largest commercial transformer models like OpenAI's GPT, Anthropic's Clause, Google's Gemini, or Meta's Llama are exclusively decoder-only.

to the model's performance. However, my experiments showed that pre-trained models provide an essential understanding of code syntax and arithmetic.

Some of these pre-trained models showed surprisingly accurate (albeit imperfect) performance in SVG-to-markup translation, even without fine-tuning.

Model size Machine learning models can be characterized by the number of their trainable parameters. Several scaling law studies have found that, generally speaking, a model's capabilities grow with the number of parameters. This is not a hard rule, and it has been demonstrated that "model scale should be chosen based on the complexity of the task and the required reasoning capabilities" [69]. This raises the question: How complex is the task I am trying to solve?

Paper	Models used or created	Parameter size
Sketch2Code [16]	Deeplab on Xception	23M
Pix2Code [14]	Pix2Code	109M
Donut [18]	Donut	143M
Pix2Struct [17]	Pix2Struct	282M, 1.3B
Frontend Diffusion [15]	Claude 3.5 Sonnet (20241022)	Unknown
UIPilot [21]	Pix2Struct 1.3B	1.3B
	GPT-4V	Unknown
ViCT [19]	GPT-2	1.5B
	Llama	7B, 13B, 33B, 65B*
	ViT	86M, 307M, 632M*

Table 6.1: Some papers discussed in the Related work section 2.2 with the parameter count of their underlying model(s). M = million, B = billion.

* Not clear from the paper which model size was used.

Table 6.1 shows the parameter count of models used by some of the papers mentioned in the Related work section 2.2, for reference. No clear patterns emerge. There are several orders of magnitude of difference in model sizes. The 10^7 magnitude appears to be the lower bound.

Besides the model size, the training budget (total FLOPs) and the training data size directly influence the performance. These three quantities are not independent of each other. As discussed in section 4.2, the training data size corresponds to an ideal parameter size and vice versa. The training cost is affected by both the parameter count and the training data size.

As discussed, training data availability is practically limitless. The training budget did, however, create a hard constraint. State-of-the-art GPUs required to train large models are costly to rent (see Appendix A.7 for a breakdown of financial costs). This put an indirect limit on the model size and the training data size. TR5

Another limiting factor of the model size was the memory available in a single GPU. In my case, a limit of 96GB video RAM was set by the highest-class GPU I could access through cloud rental services. While multi-GPU training is certainly an option, this further complicates training and increases cost. Unsloth [70], a high-performance fine-tuning tool I relied on to train the larger models, does not support multi-GPU training. This limited the trainable model size and the maximum input sequence length.

Some fine-tuning strategies like LoRA (see section 6.2.2) train just a small subset of the original model's weights. This reduces the resource requirements both in terms of memory and total FLOPs.

My final strategy for deciding on the model size was to start with smaller models. Then, if the performance was not satisfactory, progressively move to a model with a larger parameter size. For this reason, I preferred model architectures that came in gradually increasing size variants.

Maximum sequence length The original transformer's training and inference computational requirements grow quadratically with the input sequence length. It's because of the self-attention mechanism: each input token attends to the rest of the input tokens, creating a pairwise attention matrix. This limits the practical size of the input, also referred to as *context length.* TR4

Significant research has been done to reduce the quadratic complexity. Some new architectures sparsify the attention matrix according to a fixed or learnable pattern, reducing resource requirements and trading it off for model quality. Some others use novel techniques such as neural memory, downsampling, or low-rank decomposition with a similar trade-off. For a comprehensive survey, see Tay et al. [71]. Other techniques, such as FlashAttention [72] or PagedAttention [73], apply hardware optimization on training and inference without sacrificing full attention. Beyond transformers, there are several proposals for entirely novel architectures, such as Mamba [74] or Hyena [75], that address the quadratic scaling issue. Nevertheless, the transformer remains the basis of the best-performing state-of-the-art models, and input sequence length remains a hard limit, albeit expanding.

Ultimately, my training data came from the public web, which determined the input sequence length. A web page and its corresponding vector image need to be treated holistically. They cannot be segmented into smaller components at training time, such as sentences in a translation dataset. In the case of the encoder-decoder architecture, the input is the complete, tokenized SVG file (or files when training with several screen resolutions). When training a decoder-only model, both the SVG file(s) and the corresponding markup (HTML+CSS) need to fit into the maximum input sequence. In Chapter 4, I described two data augmentation techniques to obtain shorter page-image pairs from a public webpage. These techniques have practical lower bounds on how much they can reduce the data size.

Figure 6.1 is illustrative when estimating the ideal context size for training on public websites. The datasets, of course, can be filtered based on their item size. However, the perfect model architecture would use all of the crawled data.

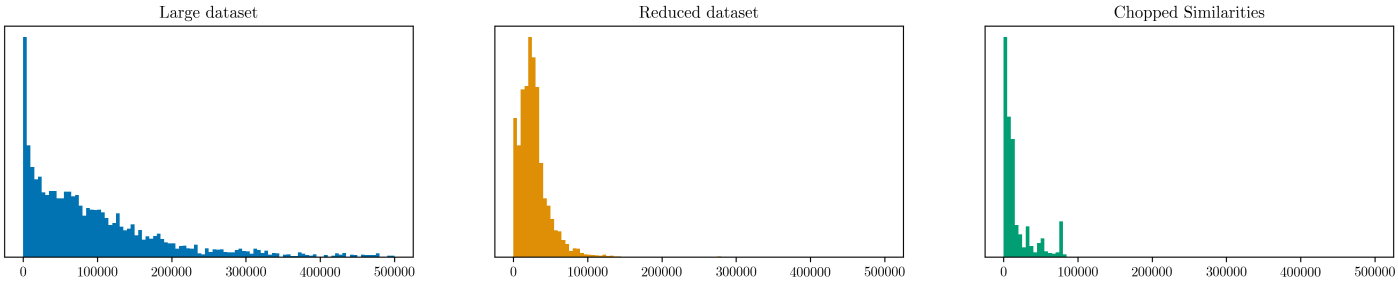


Figure 6.1: The distribution of token sizes of dataset items on all three datasets based on public websites (see section 4.4). Here, an item means a single resolution (mobile) SVG concatenated with the corresponding markup (HTML+CSS). Google's BERT multilingual base model tokenizer was used to calculate token counts. The histogram is based on a sample of 5,000 random items from each dataset.

Tokenizer Pre-trained models come with a corresponding tokenizer. During training and inference, input byte sequences are mapped to sequences of integers that are typically shorter than the input sequence in bytes. The mapping is done using a large pre-computed vocabulary based on the training dataset's frequency patterns. Pre-trained models and tokenizers are, therefore, tightly coupled. Not all tokenizers are universal in the sense that they can encode an arbitrary byte string. For example, the tokenizer of T5 [76], a popular encoder-decoder transformer, lacks special characters common in markup. This can be solved by adding new tokens to the tokenizer before fine-tuning and resizing the token embeddings. Still, since the model "sees" tokens and not individual bytes, the tokenizer's vocabulary affects the model's learning capabilities.

Since my typical input has a very different frequency pattern than arbitrary English text, I briefly explored building syntax-specific tokenizers. Due to scope and length limitations, I abandoned that idea[2].

License While most available models have permissive licenses for research purposes, I still had to consider the legal aspect. This was especially important because I am a European Union resident, and some models restrict EU usage.

[2]A basic proof-of-concept was published in the experiment repository at https://github.com/tcz/rb-experiments/blob/main/misc/syntax-aware-tokenizer-tests.ipynb

Framework availability The popularity of transformer-based machine-learning models inspired a vast array of open-source tools. Among these, HuggingFace's Transformer library [77] has emerged as a de facto standard for fine-tuning large, transformer-based models. It takes advantage of HuggingFace's hosted model and dataset library, making it very convenient to work with. Due to its prominence and community support, model availability on HuggingFace became an important factor when choosing models to pre-train or fine-tune.

Due to budget limitations, efficient training was important, especially for larger models. I chose to fine-tune larger models using Unsloth [70], an efficient fine-tuning framework. This further limited the set of available models.

Name	Sizes	Max input size*	License	Year
Encoder-decoder models				
DistilBERT [78]	66M	512	Apache 2.0	2019
XLNet [79]	110M, 340M	N/A	Apache 2.0	2019
★ T5 [76]	60M, 220M, 770M, 2.8B, 11B	512	Apache 2.0	2019
Longformer [80]	149M, 435M	4096	Apache 2.0	2020
Reformer [81]	149M	N/A	MIT**	2020
Routing transformer [82]	N/A	8192	Apache 2.0	2020
T0pp [83]	3B, 11B	1024	Apache 2.0	2021
★ LongT5 [84]	220M, 770M, 3B	16,384	Apache 2.0	2021
CodeT5 [85]	60M, 220M, 770M	512	BSD 3-clause	2021
EdgeFormer [86]	8.6M, 9.4M	N/A	MIT	2022
CodeT5+ [87]	220M, 770M, 2B, 6B, 16B	768	BSD 3-clause	2023
BigBird [88]	128M, 360M	4096	Apache 2.0	2023
Decoder-only models				
CodeLlama [89]	7B, 13B, 34B	16,000	Llama 2 Community	2023
Codestral Mamba [39]	7B	256,000	Apache 2.0	2024
Codestral [39]	22B	32,000	Mistral AI Non-Production	2024
★ Llama 3.2 [90]	1B, 3B, 11B, 90B	128,000	Llama 3.2 Community	2024
Qwen 2.5 [91]	0.5B, 1.5B, 3B, 7B, 14B, 32B, 72B	262,144	Qwen	2024

Table 6.2: Some of the models that were considered for training. The ★ symbol indicates the chosen models. Not all models have a maximum input size listed because it may depend on the implementation, or a limit does not apply. Concrete implementations were unavailable for the Routing transformer. M = million, B = billion.

* Maximum input length is not a theoretical hard limit for some models. It may indicate a maximum sequence length for a given pre-trained model or quality deterioration beyond the limit.

** Not an official implementation.

Table 6.2 shows some of the models I considered. While I did train some Codestral and CodeLlama models, I settled on the following.

T5 family (T5 and LongT5) [76, 84] For training on simple synthetic data, T5 and its expanded context-size younger brother, LongT5, proved appropriate choices. T5 (small variant, 60M) had a small context window, but some synthetic datasets did not require a

larger window. T5 was much faster to train than larger models and performed well as a cost-effective proof of concept on simple data. For longer inputs, LongT5 offered a drop-in replacement that worked with the existing T5 experiment implementation with a few changes. T5 and LongT5 are variants of the same family. LongT5 has a much larger maximum input sequence length, fitting even a large number of items of the "Large" public web dataset.

Other encoder-decoder models may have appeared more attractive on paper, but the wide range of model sizes in the same family made practical sense given my limited training budget. No other architecture offered the flexibility to scale from short context lengths and 60M parameters to longer contexts and up to 3B parameters. The model was available on HuggingFace, and the licensing terms posed no issues. The sole downside of the T5 model family was the tokenizer: its vocabulary lacked some special markup characters, so it had to be extended before training.

Llama 3.2 [90] As discussed previously, in recent years, encoder-decoder models have fallen behind decoder-only models in capabilities. This is mainly due to their popularity in commercial products. The experiments on the T5 family discovered its limitations when trained on the public web datasets. I had to choose a more powerful base model to train a capable model on those datasets.

Due to resource and budget constraints, I limited the surveyed models to those available for the Unsloth fine-tuning framework. Unsloth claims a significant training time speed-up and lower memory requirements than training on out-of-the-box implementations. Of the models considered, Llama 3.2 stood out as a modern, scalable option with a large context window, reasonable training time, and acceptable licensing terms.

6.2 Training

Over six months, I have completed 45 documented training experiments and approximately two dozen undocumented ones. The shortest training took ≈ 1 hour, the longest took 393 hours, with an average of 38 hours, and 1,694 total training hours. This excludes time spent setting up the environment and pre-processing data files. Figure 6.2 shows the training time of each documented experiment.

The documented experiments were released under the Apache 2.0 License at https://github.com/tcz/rb-experiments TR7

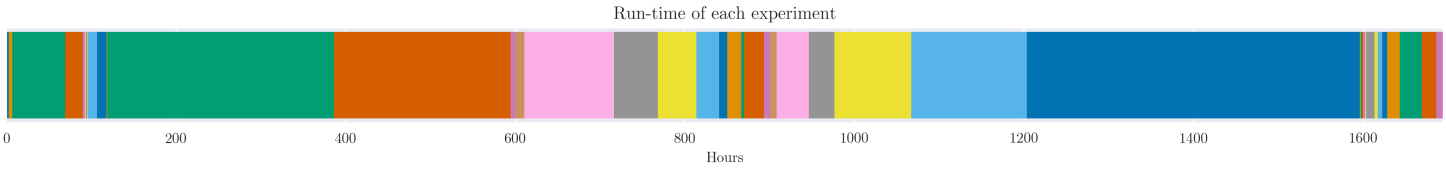


Figure 6.2: Training run-time of each documented experiment in hours. The run-time excludes setting up the environment, downloading, and pre-processing data files.

Hardware Nvidia GPUs have become a de facto standard for training modern machine learning models. As a result of Nvidia's early investment in CUDA, their GPU programming platform, nearly all major machine learning frameworks have built complex dependencies on their software and hardware.

All my training experiments were done on Nvidia H100 GPUs rented from the Vast.ai [92] cloud GPU marketplace. Vast.ai lets users rent on-demand GPU time on state-of-the-art hardware from third parties at a competitive hourly price. It uses container-based virtualization and lets users deploy pre-built Docker images on the GPU machines of their partners. They leverage NVIDIA's Container Toolkit, which passes through direct driver access to the container. Hosts create direct ports through NAT[3] so that clients get direct secure shell access to the container. Vast.ai's pricing is time-based, and the container may be terminated when no longer needed.

When possible, I rented the higher memory instances of 96GB Video RAM (over the cheaper 80GB version) because even with synthetic data, out-of-memory issues became a problem. Techniques such as gradient checkpointing (described below) may address these issues. However, many of these optimization techniques trade memory for training time. Therefore, the optimal use of financial resources often seemed to be renting the fastest, highest-memory-capable GPU available.

While multi-GPU instances are available on Vast.ai up to 8 GPUs per machine, I opted not to use multi-GPU training due to its complexity, lack of framework support, and increased rental costs while the GPUs are not utilized (e.g., installing dependencies, downloading and pre-processing data).

Software The models I trained were built on the PyTorch framework [93, 94].

HuggingFace has emerged as the standard for modern, open-source learning software tooling. HuggingFace's various supporting libraries are partially built on PyTorch[94] and implement a user-friendly, higher-level framework to train large models with modern techniques. Of these libraries, I made extensive use of Datasets [95] to create, manage, and publish datasets; Transformers [77] to reuse pre-built open building blocks such as

[3]Network Address Translation

tokenizers, models, and training glue code; Evaluate [96] for metric evaluation during and after training; PEFT[4] for efficient fine-tuning; Bitsandbytes[5] for memory-efficient quantization; TRL[6] for reinforcement learning.

Unsloth [70] is a software tool built on the HuggingFace software stack to improve resource use when training large, transformer-based models. It uses an array of performance optimization techniques I describe in section 6.2.2. I used Unsloth[70] to reduce the resource requirements and, consequently, the costs of training the larger Llama models.

During and after training, the model's output was evaluated on the validation and test datasets. This involved orchestrating a headless browser that rendered the predicted and target HTML+CSS code and compared their screen captures. I used the Python variant of Playwright[7] for this task.

I relied on Jupyter Notebook [97], a popular form of literate programming [98], to run experiments. I collected and analyzed metrics using Tensorboard [99, 9.1 TensorBoard].

For running long inference experiments on a trained model, I relied on vLLM [73], an efficient LLM inference engine. TR6

Evaluation During training, the model produces a token-level cross-entropy loss at each logged training step. This metric does not give useful insights into the model's performance. It merely indicates that the model is still learning.

To monitor the actual performance of the model, after each N steps, the model was evaluated on the accuracy metrics discussed in Chapter 5, LPIPS, and MSPS. The model generates markup, which must be rendered before comparing it with an expected output. At each evaluation step, a headless browser engine (see previous paragraph) was executed, screenshots of the predicted and expected markups were taken, and then compared to produce MSPS and LPIPS scores. The browser engine's viewport was set to one of the predefined screen sizes (e.g., mobile, desktop), and only the viewport contents are evaluated (unlike during dataset building).

This is a relatively slow process compared to traditional sequence-to-sequence evaluation metrics like BLEU. I limited the mid-training evaluation to just a few (usually four) instances to save time and resources, as described in section 4.3.

[4]https://github.com/huggingface/peft
[5]https://github.com/bitsandbytes-foundation/bitsandbytes
[6]https://github.com/huggingface/trl
[7]https://playwright.dev/

Hyperparameters The available models pre-determined the architectural hyperparameters, such as the number of layers or heads, activation function, etc. Training-related parameters were left to be decided. Of these, batch size was often a function of memory. For the public web datasets, the batch size had to be reduced to as low as 1 or 2 due to their size and the quadratic memory scaling problem discussed in section 6.1. Gradient accumulation was used for low batch sizes. To further reduce memory use, the parameter precision was reduced to 16 bits for the T5 family and down to 4 bits for Llama.

For other training hyperparameters, such as weight decay, warmup steps, learning rate scheduling, optimizer, etc., I used sensible defaults from other papers or as suggested by the frameworks' documentation, since determining these with grid search or other, more sophisticated methods would have further increased the project's costs.

Bugs During model training, many unpleasant and frustrating software bugs occurred in the libraries and frameworks I relied on.

Transformer models, especially large language models, are at the center of a very active and fast-moving research area. This, in turn, causes the related software tools to evolve at a breakneck pace. For instance, in the first four months of 2025, HuggingFace's Transformer model released 13 new versions, and contributors made 2,057,470 source code line additions and 217,692 deletions. In April 2025, there were over 1,000 open issues on their GitHub page. The pressure to keep up with the latest research papers and techniques causes considerable instability in these projects. This made working with these tools challenging, and I experienced frequent, often weeks-long setbacks due to bugs, poor documentation, and unimplemented features.

6.2.1 T5 family

In this section, I will showcase a few select experiments on the T5 family.

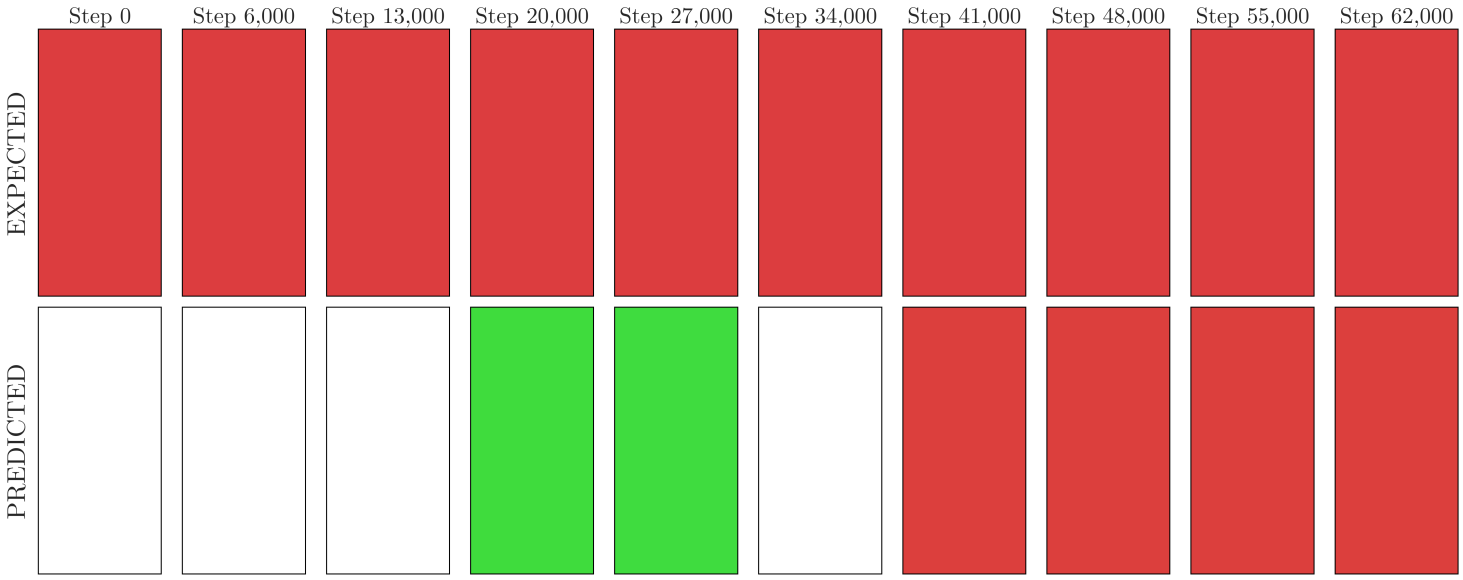
As described in section 4.3, a set of simple synthetic datasets was created to prove that transformers can learn to translate simple web features such as colors, text, and size. TG2

The number of epochs was empirically determined and set at 10 to ensure the loss converges (unless otherwise indicated). The model with the best validation accuracy scores at each evaluation step was saved to avoid overfitting. The models were evaluated every 1,000th step on the validation set. For the synthetic datasets, the validation and test sets were partitions of the entire dataset, four items and 100 items, respectively. For the public web datasets, the validation set was the "Small Validation" set described in Chapter 4. The evaluation included a special, 0th step on an untrained model, or, in the case of fine-tuning, the base model. After the training, the best model was evaluated on the test set. The same procedure was followed for all the T5-family experiments listed here.

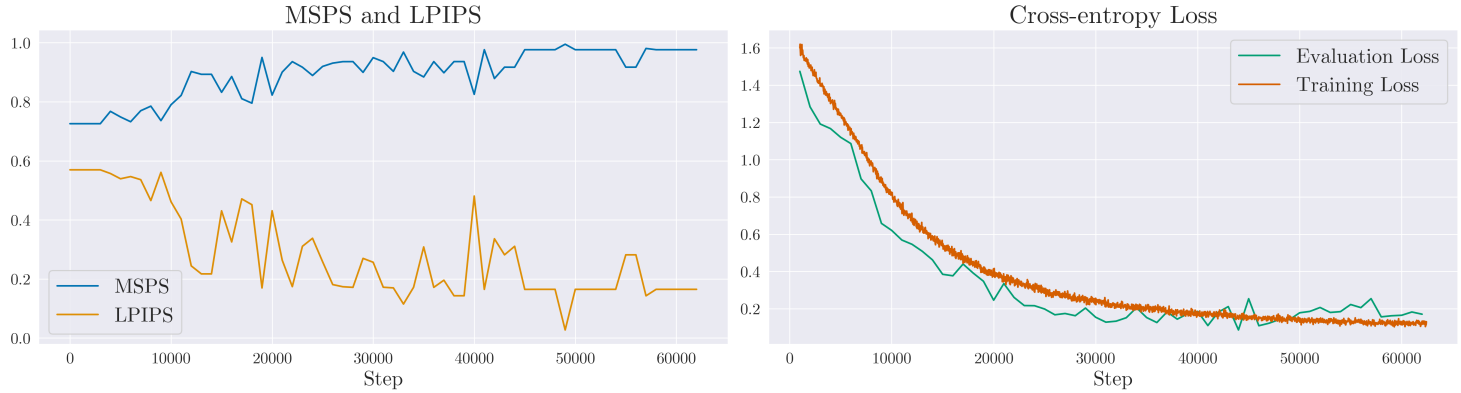
Lower precision (16 bits) was applied to reduce memory use. Learning rate was 2×10^{-5} with no warmup. Weight decay was set at 0.01 and batch size at 16 unless indicated otherwise. The stock T5 tokenizer (based on SentencePiece [100]) had to be extended with new tokens because this tokenizer was built for natural language processing, and so it lacks certain tokens and adds extra white space after each token at decoding.

Synthetic "Color" dataset, pre-training In this experiment, I tested training a T5 model initialized with random weights on the simplest synthetic dataset, "Color", described in section 4.3. The model must learn to match the repeating SVG syntax with the corresponding HTML+CSS parts. It must also learn to convert between the decimal RGB color representation used in SVG and the hexadecimal syntax used in the original synthetic HTML, e.g., `rgb(207, 32, 23)` to `#cf2017`. Ordinarily, this would require arithmetic skills, but it's likely that the model instead simply learns to memorize corresponding digit tokens.

Figure 6.3 shows the evolution of the training.



(a) Evolution of experiment predictions



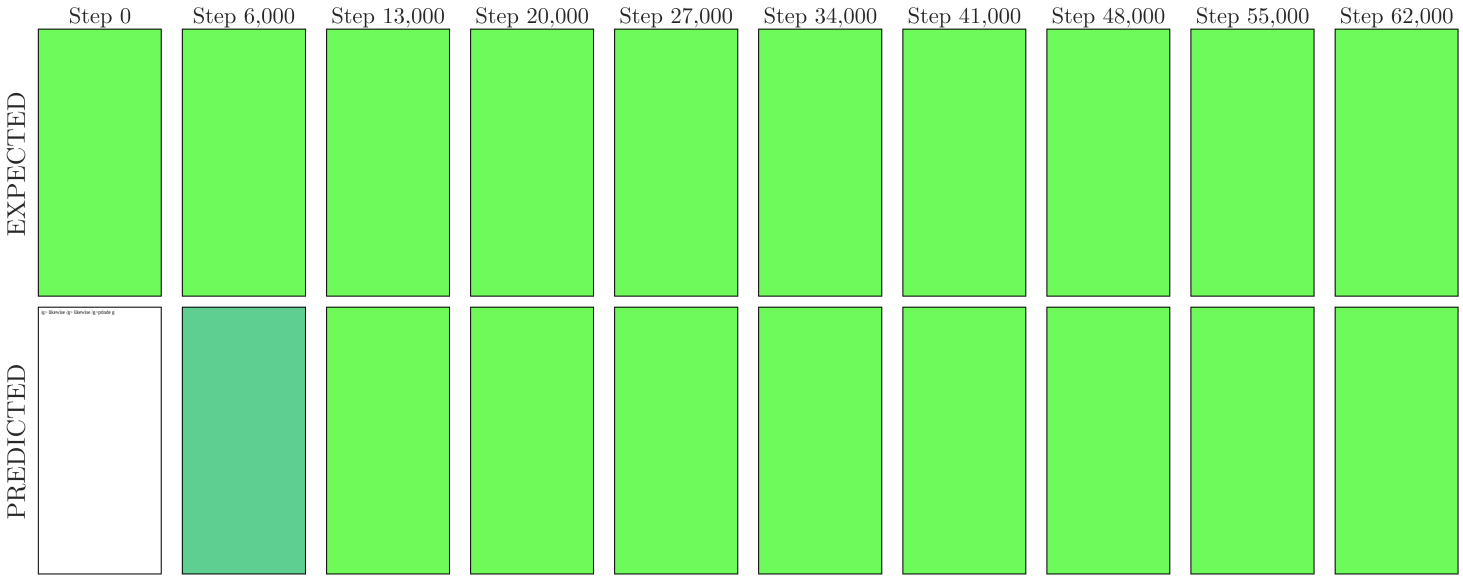
(b) Accuracy metrics and cross-entropy loss

Figure 6.3: T5 pre-training on the "Color" dataset. The upper figure shows the evolution of experiment predictions using a single sample from the validation dataset, rendered in a browser engine, compared to the expected ground truth. The lower left figure shows the evolution of the accuracy metrics on the validation dataset as defined in Chapter 5. The lower right figure shows the cross-entropy loss on the validation (unseen) and the training datasets.

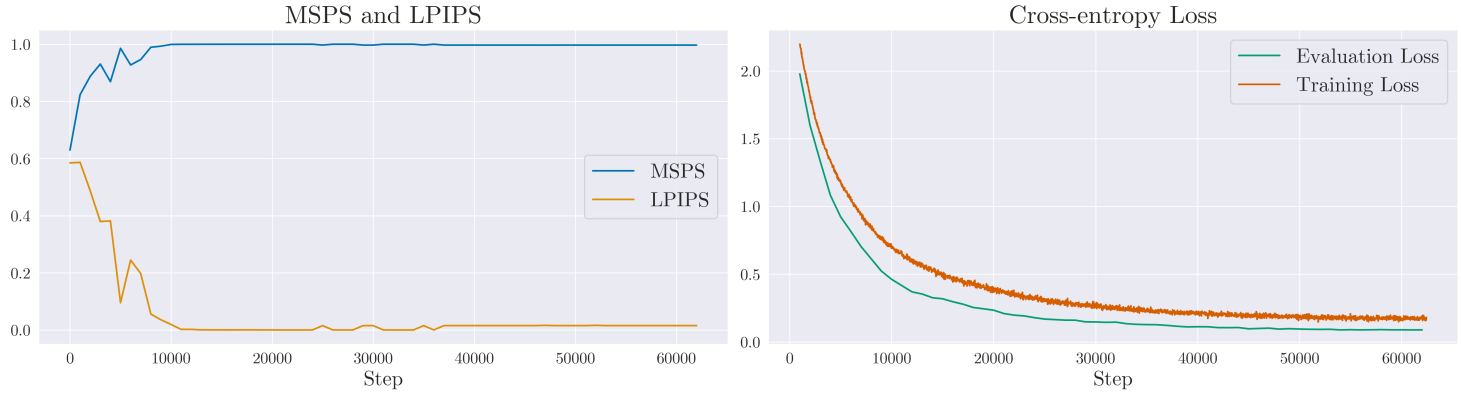
The training took 1 hour 12 minutes and 9.16×10^{16} FLOPs. The best model was saved at 49,000 steps (MSPS: 0.9949, LPIPS: 0.0274).

The final accuracy scores from the test set were **0.9642** MSPS and **0.0986** LPIPS. These scores were disappointing given the simplicity of the dataset, but the model demonstrated that it can learn the tasks defined above.

Synthetic "Color" dataset, fine-tuning This experiment repeats the previous one, but instead of starting a model with random weights, it fine-tunes the pre-trained base model. All else is equal.



(a) Evolution of experiment predictions



(b) Accuracy metrics and cross-entropy loss

Figure 6.4: T5 fine-tuning on the "Color" dataset. See the caption of Figure 6.3 for more information.

As apparent from Figure 6.4, the model reaches perfect accuracy on the validation set much faster than the previously untrained one. This indicates that the new model takes advantage of knowledge previously instilled into the base model. This suggests that, generally, we can expect better results from fine-tuning than pre-training. This is remarkable given that T5 was trained on natural language sentences, explicitly avoiding program code. Nevertheless, the model seems to have learned useful representations that help it learn faster in this domain and likely no longer simply memorizes hexadecimal representations of decimal integers.

The best model was saved at 24,000 steps (MSPS: $\approx$ 1.0000, LPIPS: 0.0001). The final accuracy scores were **0.9935** MSPS and **0.0285** LPIPS, markedly better than the previous experiment.

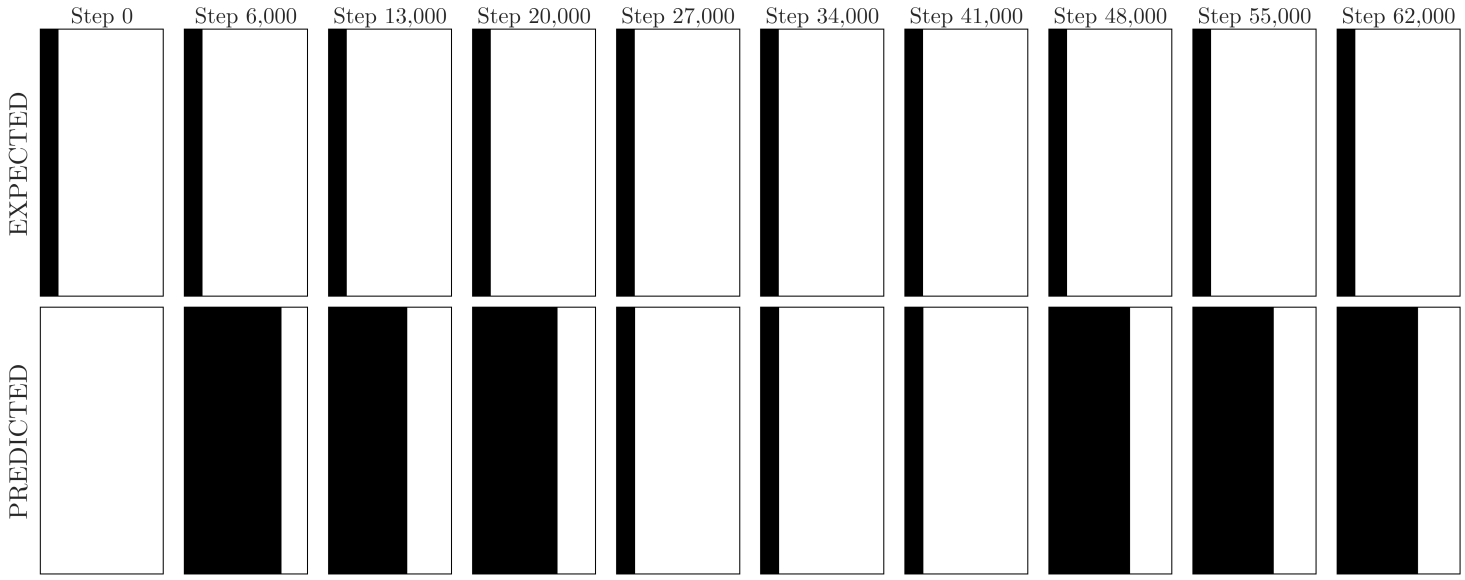
Synthetic "Size" dataset, pre-training Transformer models can learn arithmetic [101–103], but the T5 models are not good at math. Even the larger pre-trained models fail to answer very simple arithmetic questions, see Table 6.3. As discussed in section 4.3, calculating the layout of a web page involves many simple arithmetic operations, and so does converting a vector image to HTML.

Input	Output
Question: What is the capital of Italy?\nAnswer:	turin ✗
Question: When was the Battle of Waterloo?\nAnswer:	1815 ✓
Question: What is the chemical name of table salt?\nAnswer:	sodium chloride ✓
Question: What force pulls objects toward the Earth?\nAnswer:	gravity ✓
Question: What is 1 + 1?\nAnswer:	1 ✗
Question: What is 25 - 3?\nAnswer:	0 ✗
Question: What is 2 * 5?\nAnswer:	5 ✗
Question: What is 100 / 2?\nAnswer:	100 2 ✗

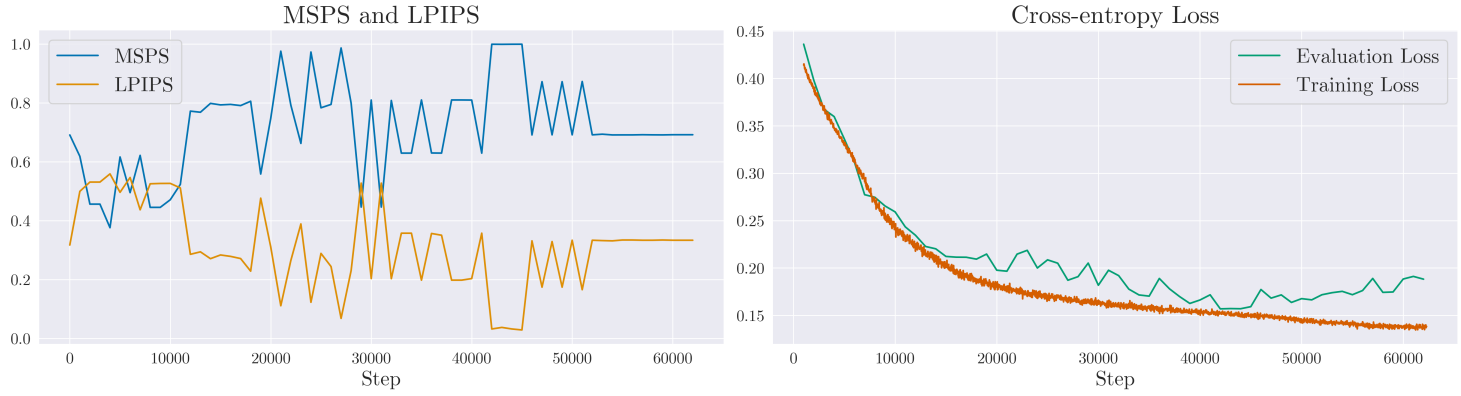
Table 6.3: A few arbitrary general knowledge and simple math questions and the answers obtained from a pre-trained T5 model. The tick and cross indicate correctness. Here, the "Large", 700M parameter variant of FLAN-T5 [104] was used, an instruction-fine-tuned family built on the original T5 models. FLAN-T5 models are better at question answering than the original T5 family. Note that the model performed equally poorly when the math questions were phrased as text (e.g., "What is twenty-five minus three?"). "\n" represents the "new line" character.

In this experiment, I tested training a T5 model with random weights on the "Size" dataset. The model must learn to convert a width value defined as absolute size (pixels) in the SVG to a relative value (viewport percentage) in HTML, e.g., `92.28125` to `23.485%`. This can be thought of as unit conversion, which consists of a simple division.

The training took 1 hour 14 minutes and 1.04×10^{17} FLOPs.



(a) Evolution of experiment predictions



(b) Accuracy metrics and cross-entropy loss

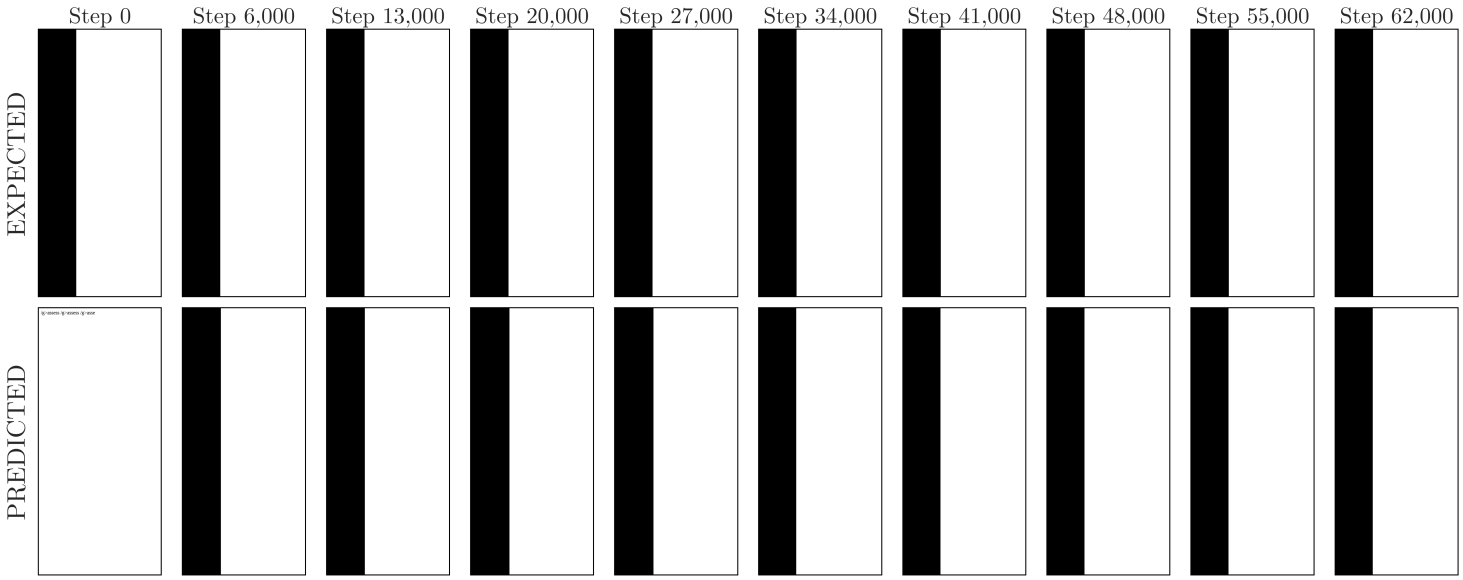
Figure 6.5: T5 pre-training on the "Size" dataset. See the caption of Figure 6.3 for more information.

Figure 6.5 tells the story of a suboptimal training. The evolution of the accuracy metrics looks jagged, likely due to the small validation set and the very simple page structure. The model gets the correct ratio of black-to-white blocks by chance. The cross-entropy loss on the validation set tends to be higher than on the training set, and it grows in later epochs, indicating overfitting. The visual prediction sample confirms this.

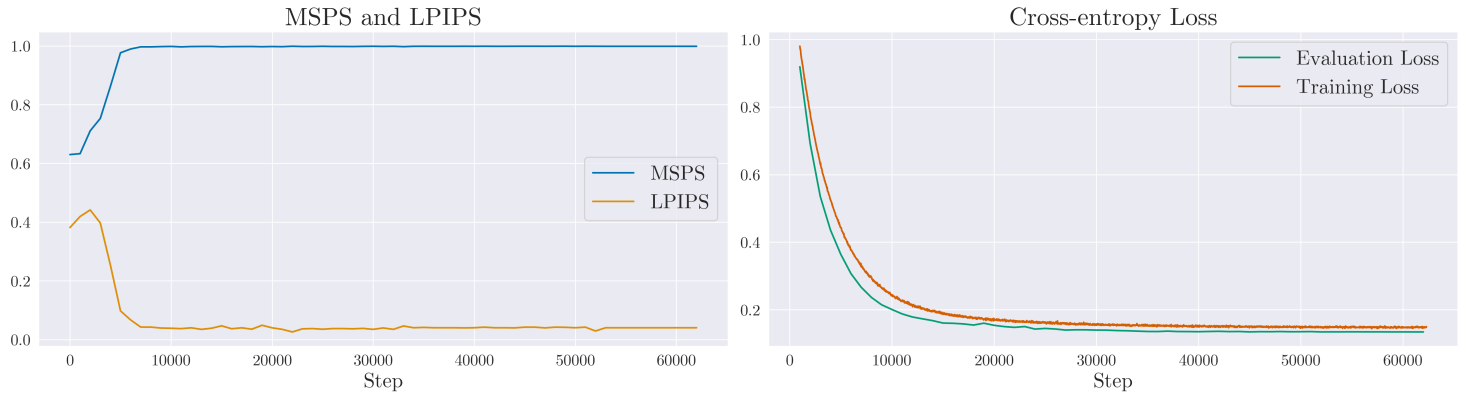
The best model was saved at 45,000 steps (MSPS: 0.9989, LPIPS: 0.0287). The final accuracy scores were **0.8754** MSPS and **0.1526** LPIPS. Disappointing results, but perhaps fine-tuning the base model could be the solution here, too.

Synthetic "Size" dataset, fine-tuning This experiment, like the one with the "Color" dataset, is a carbon copy of the previous experiment. The sole difference is that instead

of training a new model from scratch, it fine-tunes a base model that was previously pre-trained with generic natural language text.



(a) Evolution of experiment predictions



(b) Accuracy metrics and cross-entropy loss

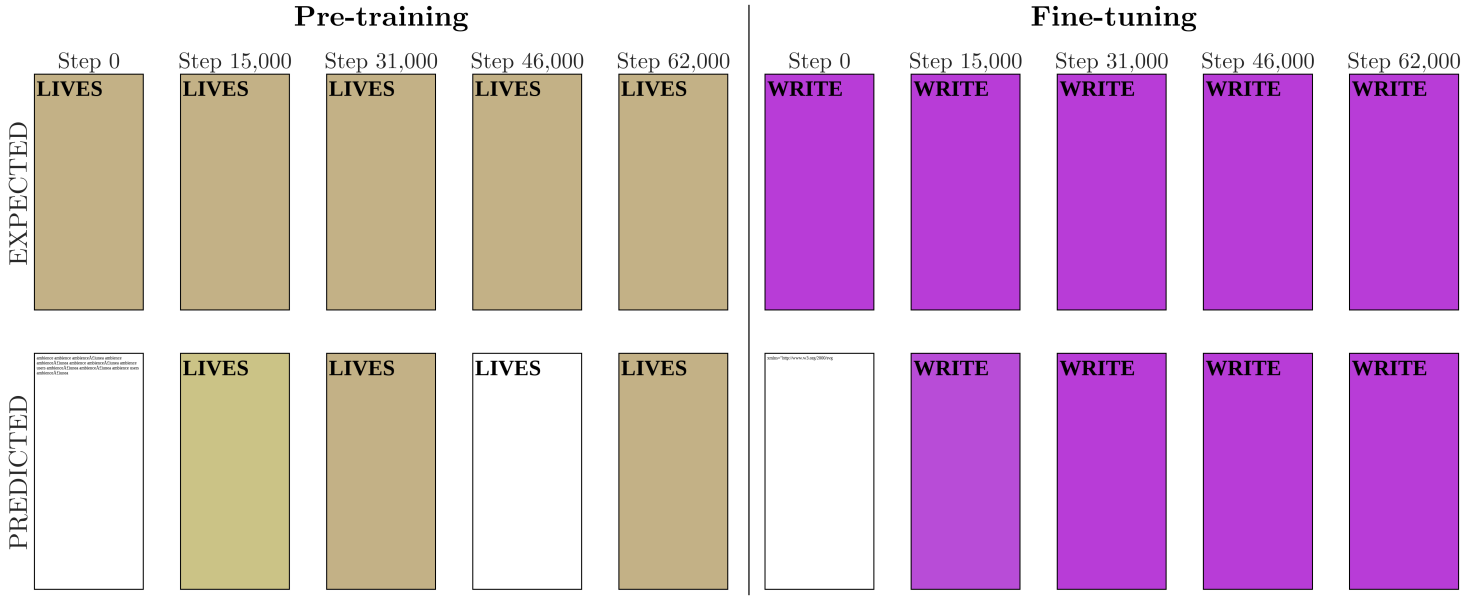
Figure 6.6: T5 fine-tuning on the "Size" dataset. See the caption of Figure 6.3 for more information.

Once again, the model converges much faster and stays stable in later epochs. This suggests transfer learning from a completely different domain. However, it remains doubtful that the model learned any practical math skills. It might have simply found patterns in the division. It works for this very simple dataset, but whether it can deal with a large set of diverse layouts with more complex calculations is dubious.

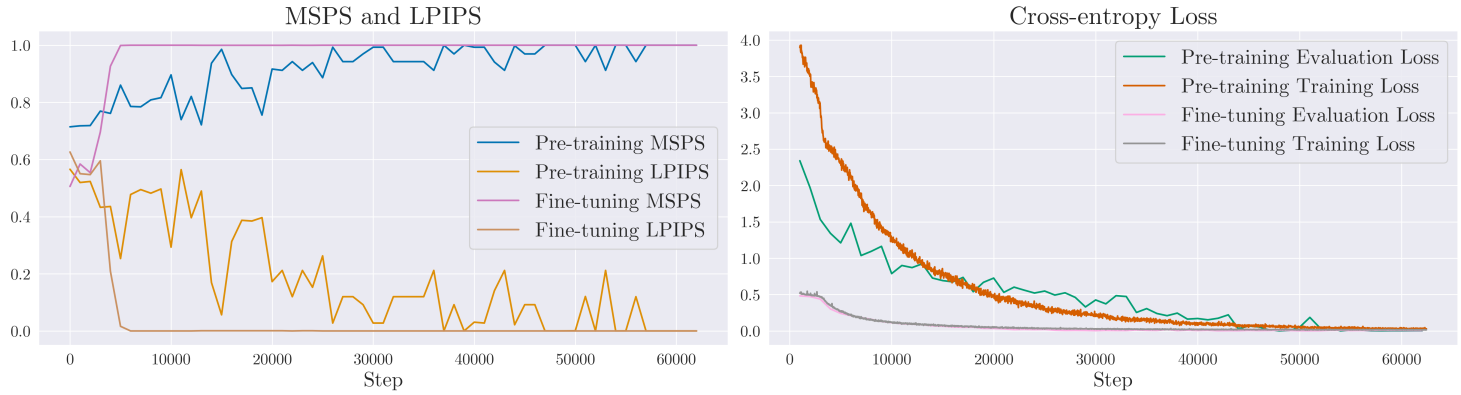
Still, the final test set score was convincing: **0.9985** MSPS and **0.0318** LPIPS. The best model was saved at 22,000 steps (MSPS: 0.9988, LPIPS: 0.0259).

Synthetic "Color+Text" dataset, pre-training and fine-tuning The next experiments were performed on the "Color+Text" dataset. This is arguably a less interesting increment, since the dataset does not add substantial difficulty over the "Color" dataset. What makes these interesting is that the input no longer fits into the 512-token limit of T5, so the model was changed to LongT5, which supports inputs of over 16,000 tokens. This also brought a nearly four-fold increase in the number of parameters: from 60M to 220M, the smallest available in each model.

Figure 6.7 combines the pre-training and fine-tuning experiments.



(a) Combined evolution of experiment predictions



(b) Combined accuracy metrics and cross-entropy loss

Figure 6.7: LongT5 pre-training and fine-tuning on the "Color+Text" dataset. See the caption of Figure 6.3 for more information.

The pattern is familiar: fine-tuning converges much faster and reaches a stable, perfect validation loss. Due to the larger model, training took longer, approximately 5 hours and

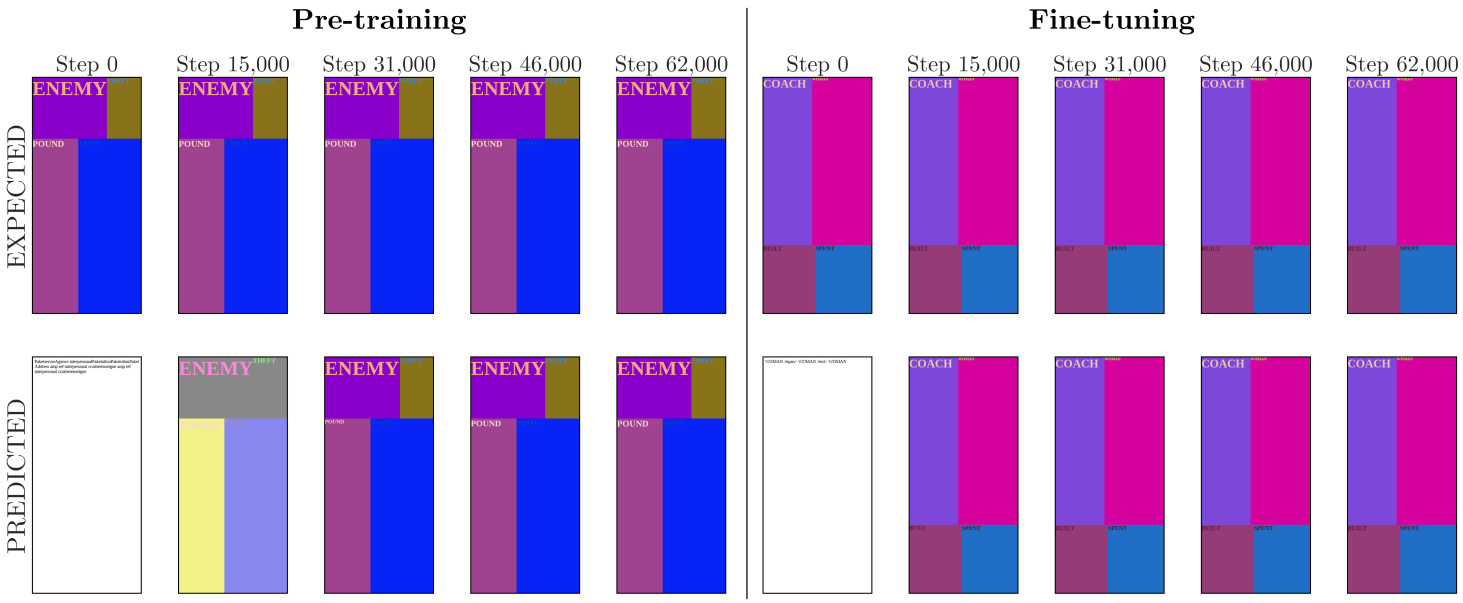
30 minutes and 8.64×10^{17} FLOPs in both cases. Evidently, the fine-tuning could have stopped early after stabilizing at perfect loss. The best models were saved at 37,000 and 7,000 steps for pre-training and fine-tuning, respectively.

The resulting test scores are **0.9725** MSPS and **0.0376** LPIPS for the pre-training and **0.9982** MSPS and **0.0109** LPIPS for the fine-tuning. This hints at higher-quality learning with the LongT5 architecture than with T5 (small). This is likely due to the higher number of parameters.

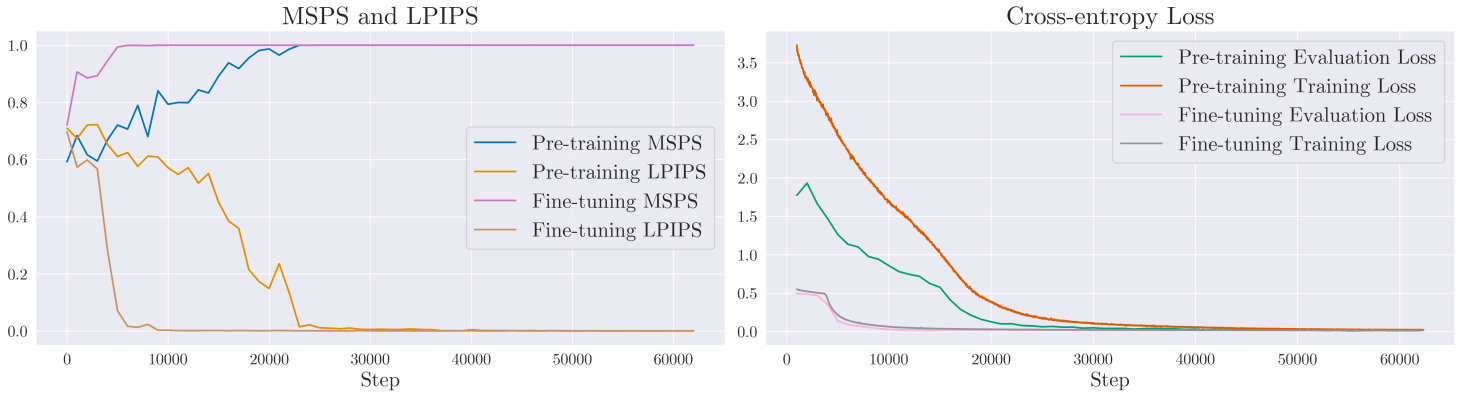
Synthetic "Color+Text+Size" dataset, pre-training and fine-tuning The most complex synthetic dataset combines features from the previous sets: color notation translation, size calculations, and text matching. The model must learn all those features at once. This is still a fairly simple, homogeneous dataset. The LongT5 model was used here as well due to the input length. Here, gradient accumulation (2 steps) and gradient checkpointing were used due to memory constraints. The former postpones updating the model's weights for some steps by collecting and aggregating gradients. This permits using smaller mini-batch sizes by creating "virtual" batches. The latter discards model activations from the forward pass and recalculates them as needed during backpropagation, trading FLOPs for memory.

Figure 6.8 shows a successful training. See results below.

	Pre-training	Fine-tuning
Training time	14h 50m	26h 18m
Training FLOPs	2.54×10^{18}	2.54×10^{18}
Best model steps	54,000	37,000
Test set MSPS	**0.9998**	**0.9999**
Test set LPIPS	**0.0047**	**0.0015**



(a) Combined evolution of experiment predictions



(b) Combined accuracy metrics and cross-entropy loss

Figure 6.8: LongT5 pre-training and fine-tuning on the "Color+Text+Size" dataset. See the caption of Figure 6.3 for more information.

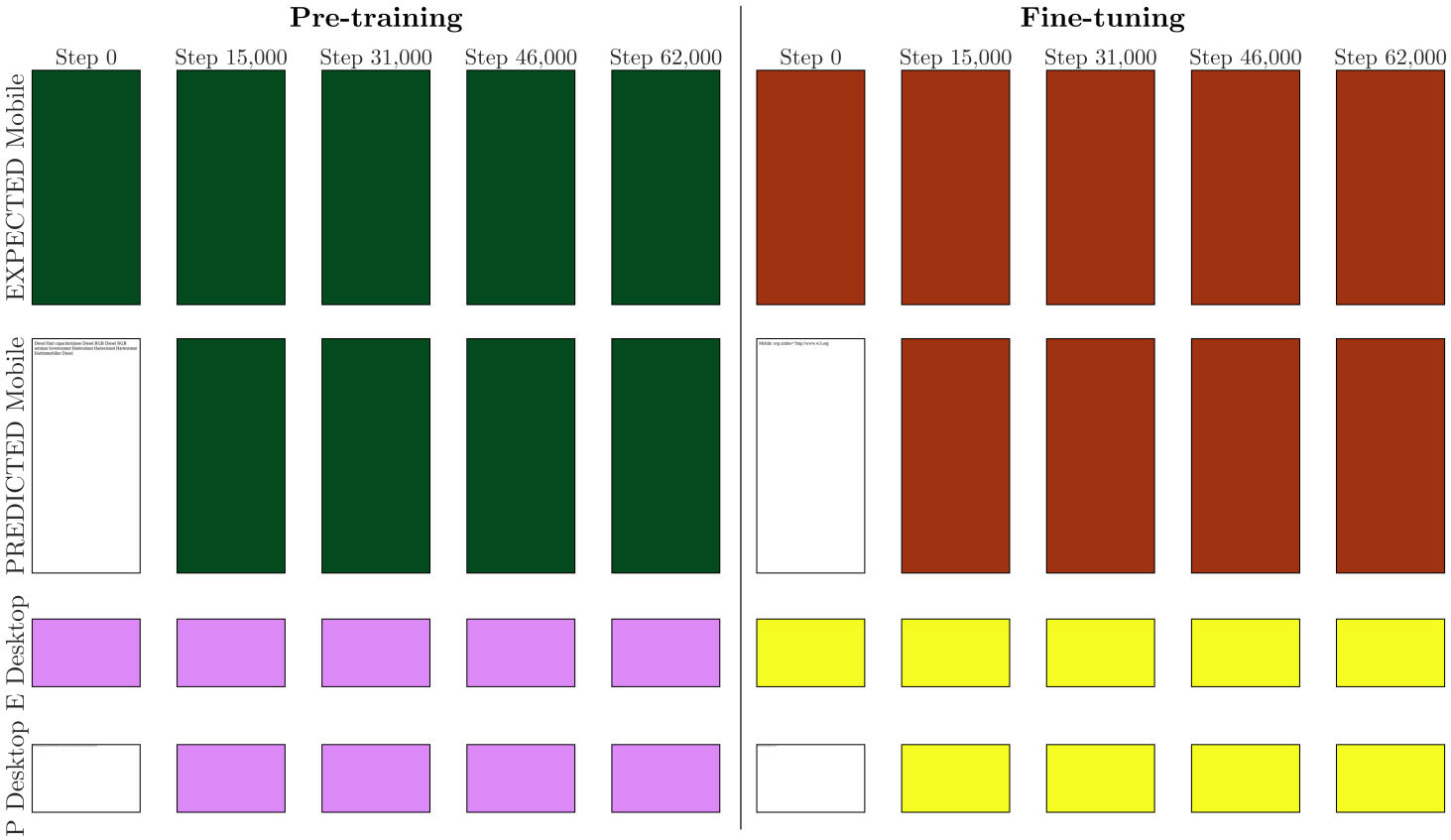
Synthetic "Color responsive" dataset, pre-training and fine-tuning We have already discussed the issue of responsive web design. The model must learn to produce a singular markup output from several different designs corresponding to distinct screen resolutions. In section 6.1, I mentioned the hypothesis that simply concatenating the input sequences representing different screen resolutions might work to produce a responsive website. I added segment separator embeddings, although I don't think it was necessary for a simple dataset like this. I suspect these embeddings (simple words describing each resolution) might improve performance for more complex designs, if the model learns (or has prior knowledge of) the syntactical meaning of "mobile", "desktop", etc.

An input instance looks like this:

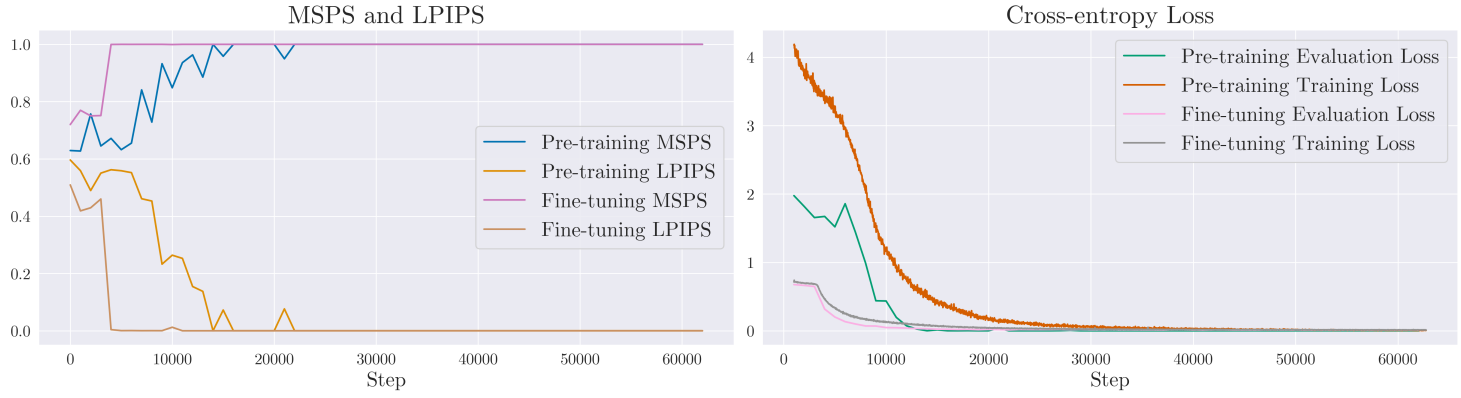
```
Mobile:

[MOBILE RESOLUTION SVG]

Desktop:

[DESKTOP RESOLUTION SVG]
```

Like the rest of the synthetic data experiments, this one too is simple and limited. It serves as a starting point to prove that the least complex version of an idea works.

Figure 6.9 demonstrates that the model was able to learn responsive features of different screen resolutions. This is hardly surprising. From the transformer model's perspective, the input is treated as a single document, and so is the output. The only potential challenge concerns the long-distance attention mapping. The strongest attentional links (the different color definitions) bridge remote positions between the source and target sequences. This might pose challenges for longer, more complex input and output sequences, especially since, for LongT5, the longer context window was made possible through a sparse attention matrix.



(a) Combined evolution of experiment predictions. Mobile and desktop expectations and predictions correspond to the same responsive web page.



(b) Combined accuracy metrics and cross-entropy loss

Figure 6.9: LongT5 pre-training and fine-tuning on the "Color Responsive" dataset. See the caption of Figure 6.3 for more information.

The training was successful. The accuracy metrics are calculated and averaged for both mobile and desktop resolutions.

	Pre-training	Fine-tuning
Training time	3h 39m	9h 59m
Training FLOPs	9.32×10^{17}	9.32×10^{17}
Best model steps	14,000	12,000
Test set MSPS	**0.9480**	**0.9981**
Test set LPIPS	**0.0989**	**0.0152**

Synthetic "Combined" dataset, fine-tuning The last experiment using synthetic data is an important one. We have proved that the model can learn to match vector image parts to corresponding markup, translate between different value representations, and do rudimentary layout math, albeit most likely through brute force. In this trial, the model must learn all that from a diverse dataset, or at the very least, more diverse than the previous datasets with repeating templates. This is not the premiere yet, but perhaps the dress rehearsal.

The "Combined" dataset contains all previous datasets (except the responsive one) plus validation and test sets in which all four synthetic datasets are represented in equal numbers of instances. Due to the size of the dataset, I decided to reduce training epochs to 2. I have previously demonstrated pre-training and fine-tuning experiments on all these datasets separately. I pointed out that fine-tuning generally converges better, typically in under two epochs. Thus, for this experiment, I only performed fine-tuning on the LongT5 Base model.

Figure 6.10 shows the evolution of the different loss metrics.

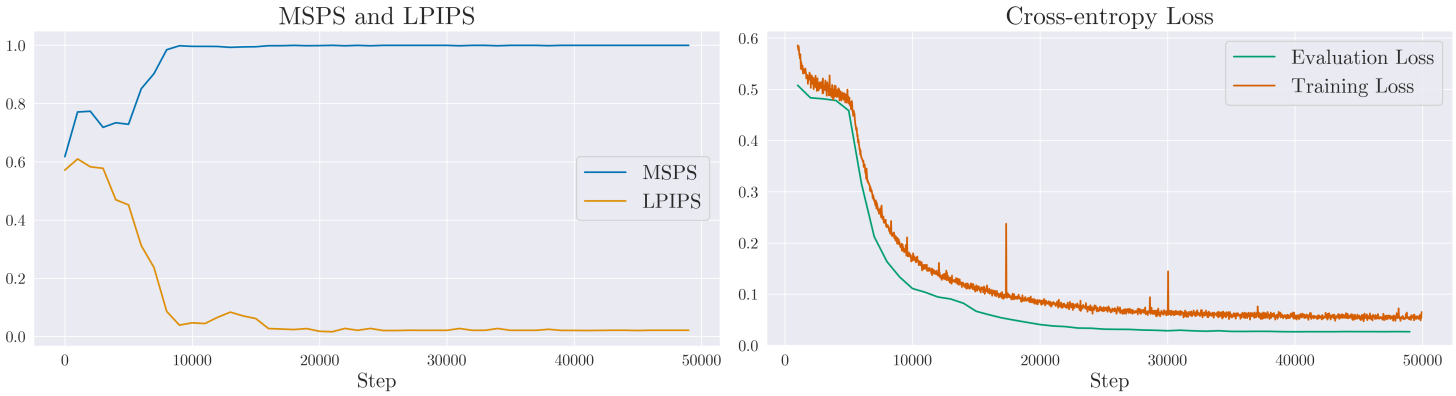


Figure 6.10: LongT5 fine-tuning on the "Combined" dataset. See the caption of Figure 6.3 for more information.

The model had no difficulty learning to translate different input types. Evidently, the training set is still far from fully heterogeneous. It contains four large blocks of simple, repeating templates. Nevertheless, the results here give confidence to move on to real-world datasets.

The training took 16 hours and 49 minutes and 1.90×10^{18} FLOPs. The best model was saved at 21,000 steps (MSPS: 0.9995, LPIPS: 0.0162). The final test set scores are: **0.9980** MSPS and **0.0212** LPIPS.

Public "Chopped" dataset, pre-training With basic concepts proven on synthetic data, it was time to move on to the training data from the public web. TG3 First, I decided to train on the "Chopped" dataset. As explained in section 4.4, this dataset contains pieces of real-world websites and their corresponding stylesheets, limiting the HTML size to 1,000 bytes. This limit was enforced not just to reduce memory use and training time, or to make sure the input fits under the model's length limit. It is just as important that this reduces the structural complexity of each data instance, making it supposedly easier to train the model.

For reasons detailed in section 4.3, the "Small Validation" dataset was used for mid-training validation. The previously mentioned issues with the pre-trained T5 family's tokenizer could no longer be patched by simply adding new tokens and resizing embeddings. There are too many unknown tokens in a public web dataset. So, I trained a new tokenizer from scratch based on SentencePiece using a few thousand items from the large public dataset (both vector and markup).

The new tokenizer meant that we could no longer rely on the pre-trained model for fine-tuning, since it was trained on a completely different tokenizer.

Even with the "Chopped" dataset, input tokens had to be limited to 2,048 to avoid GPU out-of-memory errors. Longer inputs were simply filtered, leaving a training split of about 93,000 pairs. The training ran for five epochs with gradient accumulation of 4, a batch size of 4, and a learning rate of 2×10^{-5}. It took 20 hours and 44 minutes or 5.04×10^{17} FLOPs to complete.

As Figure 6.11 demonstrates, the results were poor. Accuracy metrics stayed flat, indicating no learning. The model's output was a nonsensical jumble.

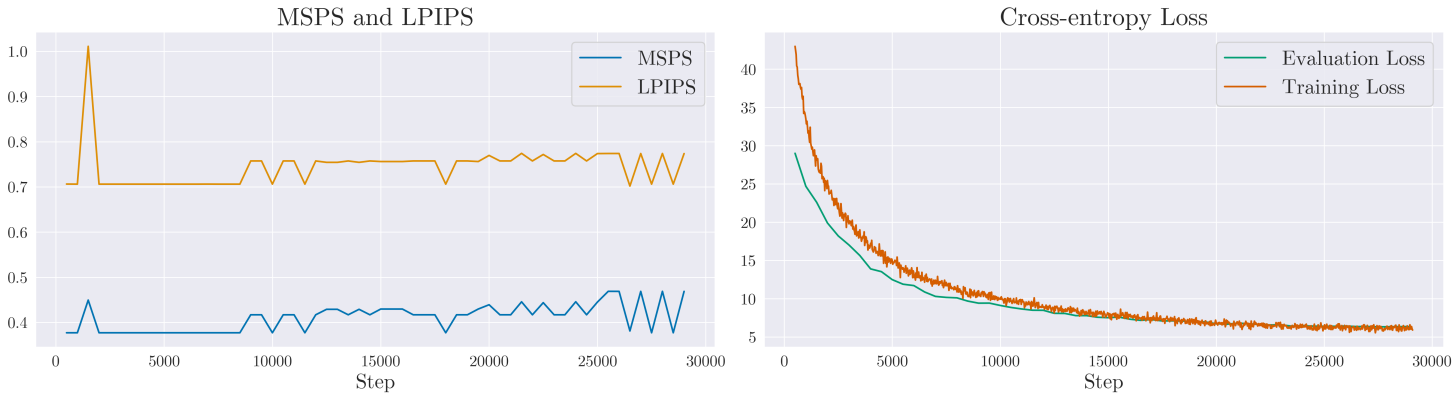


Figure 6.11: LongT5 pre-training on the "Chopped" dataset. See the caption of Figure 6.3 for more information.

The new tokenizer might affect the model's performance. Depending on how syntax gets represented by the tokenizer, the model might find it easier or harder to attend to that syntactic dependency. I repeated the "Color" pre-training experiment with the new tokenizer to discard this as a reason for the poor performance. The training converged without issues. My intuition was that the complexity of the training set had pushed the T5 family to its limits. In Table 6.3, I have hinted at the outlines of those limits. Here, it would perhaps be reasonable to try a larger variant of LongT5 before moving on. That said, the answers in Table 6.3 come from the "Large" variant of T5, and they point to a limited model performance. This family worked well to quickly and cost-effectively demonstrate the concept with simple data, but it might not be a good fit for the public web dataset.

Public "Large" dataset, pre-training Even so, before moving on from the T5 family to a more capable model, I wanted to see the model's performance when trained on the "Large" dataset. The "Chopped" dataset has some idiosyncrasies (e.g., it's very CSS-heavy) that might result in poor performance.

Figure 6.12 tells the story of another unsuccessful training. The discontinuity in the accuracy graph is due to sporadic resource-related issues with web page rendering (this was fixed in later iterations). Trained for one epoch with batch size one, gradient accumulation of 16 steps. The training took 62 hours 29 minutes or 8.19×10^{17} FLOPs.

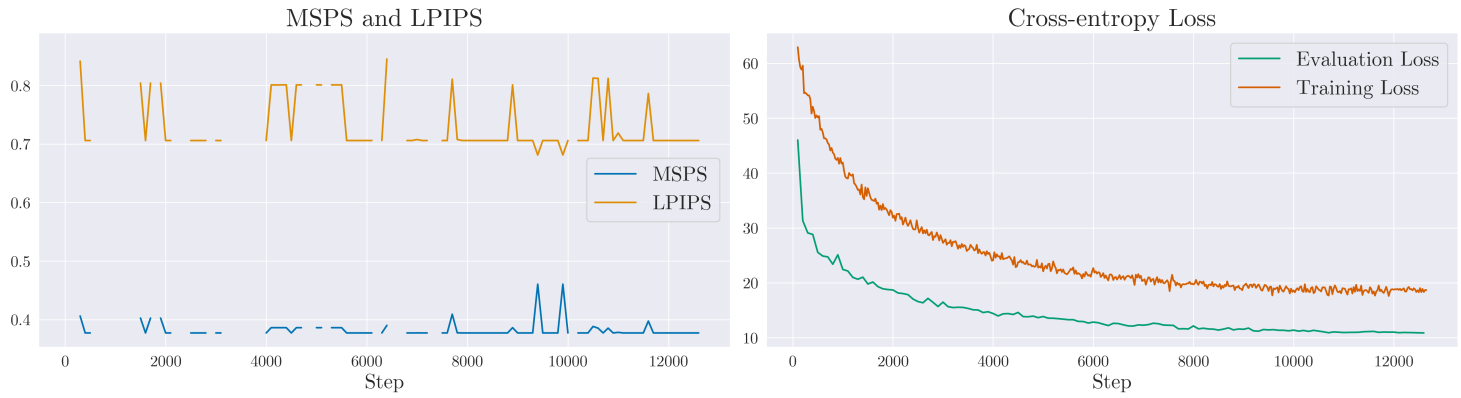


Figure 6.12: LongT5 pre-training on the "Large" dataset. See the caption of Figure 6.3 for more information.

Following another failed experiment on the T5 family, we may turn our attention to the more powerful Llama 3.2 family. TR3

6.2.2 Llama

The size difference between the "large" T5 (770M parameters) and the smallest Llama 3.2 (1B parameters) models is well under an order of magnitude. Still, the difference in

capabilities is striking. Consider Table 6.4 with the same questions as Table 6.3 (previously answered by the Large T5 variant) and a few additional, higher difficulty arithmetic questions. The basic arithmetic questions were correctly answered, and while the more difficult ones are mostly incorrect, two out of the three incorrect answers are *roughly* correct. For our objectives, rough mathematical correctness can be valuable.

Input	Output
Question: What is the capital of Italy?	Rome. ✓
Question: When was the Battle of Waterloo?	The Battle of Waterloo took place on June 18, 1815. ✓
Question: What is the chemical name of table salt?	Sodium chloride. ✓
Question: What force pulls objects toward the Earth?	Gravity. ✓
Question: What is 1 + 1?	2. ✓
Question: What is 25 - 3?	22 ✓
Question: What is 2 * 5?	10 ✓
Question: What is 100 / 2?	50 ✓
Question: What is 238 * 123?	238 * 123 = 29,166 ✗
Question: What is 5688 + 574 - 10?	5688 + 574 = 6252 6252 - 10 = 6242 ✗
Question: What is 133336 / 56?	23907.36 ✗
Question: What is 26 ^ 4?	26 ^ 4 = 456976. ✓

Table 6.4: Arbitrary general knowledge and math questions and the answers obtained from a Llama 3.2 model. The tick and cross indicate correctness. The orange cross means "incorrect but roughly correct". Here, the 1B Instruction-tuned model was used. The system prompt was "Answer the question. Prefer short answers."

Llama 3.2 is trained on a much larger number of tokens (15 trillion) than T5 (35 billion). Unlike T5's C4 training dataset, Llama's training data is also more diverse. It is multilingual and was not filtered to contain only proper sentences. It went through a more careful, model-based quality filtering, where other large language models were used to judge quality criteria. Most importantly, its training data contains code and math fed into the model through a specialized pipeline.

The fact that Llama is a decoder-only model did not noticeably impact my experiments. In practice, the self-attention in decoder-only models subsumes the function of encoder–decoder cross-attention. This explains why research on encoder-decoder architecture is largely neglected over decoder-only models.

The following experiments will demonstrate fine-tuning Llama 3.2 with our datasets. The goal is to take advantage of the underlying knowledge distilled in Llama models.

Efficient training Even the smallest Llama model with 1 billion parameters is more than four times larger than the largest model in the previous experiments (LongT5 Base). To minimize resource requirements and cost, I took advantage of the following modern, efficient fine-tuning techniques. TR5

- **Low-precision optimizer** The AdamW [105] optimizer's momentum tensors are reduced to 8 bits instead of the usual 32 bits, reducing training memory footprint.
- **Mixed precision weights** Activations and weights are stored in 16-bit floats, half of the original precision, further reducing training memory footprint.
- **Gradient checkpointing** Trading compute for memory use reduction. Activations are "forgotten" after the forward pass and are recomputed on the fly during backpropagation.
- **Gradient accumulation** Using small batch sizes, often as small as one, saves on memory use. Gradient accumulation delays weight updates until a certain number of batches have been processed.
- **Unsloth's fused kernels** The Unsloth fine-tuning framework implements efficient, fused rotary-embedding and Triton kernels that are highly optimized for GPU resource use in terms of FLOPs and are swappable with the original PyTorch implementation.
- **FlashAttention 2** By exploiting GPU-specific memory architecture and re-implementing the original transformer attention mechanism in a hardware-optimized way, FlashAttention 2 [106] provides significant speedup and memory use reduction over the original PyTorch implementation.
- **Filtering input by length** When training on public web datasets, the GPU memory limited the maximum input length. Even with all the aforementioned techniques, the dataset had to be filtered by input size (which included vector and markup together). The limit was empirically determined for each specific model size.
- **LoRA** Low-Rank Adaptation and its extensions QLoRA and rsLoRA provide the special benefit of being able to fine-tune models of this size at all on commodity hardware. Without LoRA, it would not be feasible to fine-tune models of this size (1B-90B parameters) under this project, so it deserves a section on its own. I will expand on LoRA below.

LoRA During ordinary fine-tuning, the optimizer works on all weights, and the backpropagation step may update any model parameter. Since all parameters are trainable, they all have a memory footprint. Depending on the data type, each parameter occupies 2-4 bytes. The optimizer's state may add another 2-4 bytes per parameter. This makes training a large model on a single GPU prohibitively memory-intensive.

In contrast, Low-Rank Adaptation (LoRA) [107] restricts the number of trainable parameters and freezes the rest of the parameters of the base model.

$$\Delta W = A\,B, \qquad A \in \mathbb{R}^{d_{\text{out}} \times r},\ B \in \mathbb{R}^{r \times d_{\text{in}}},\ r \ll \min(d_{\text{out}}, d_{\text{in}}) \tag{6.2.2.1}$$

$$W = W_0 + \Delta W$$

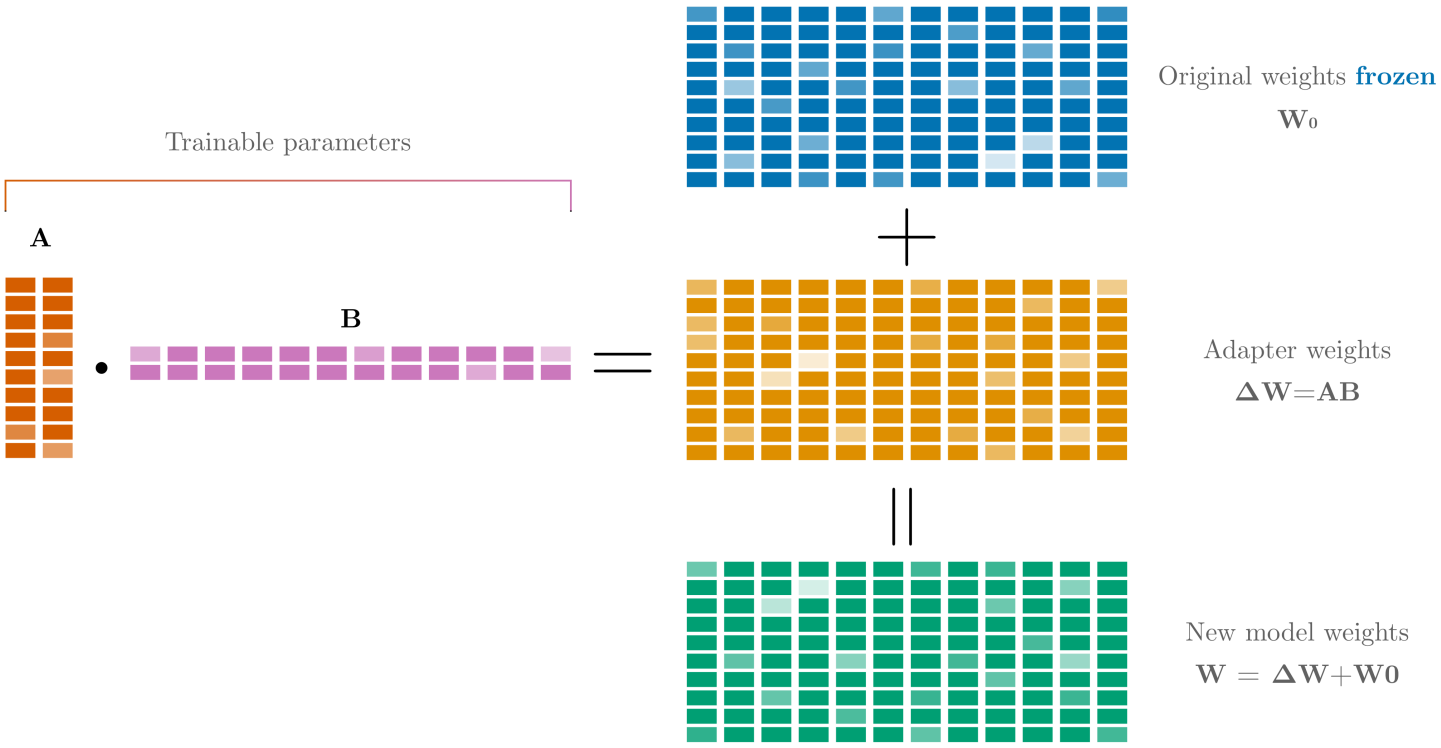


Figure 6.13: Visualization of Low-Rank Adaptation (LoRA)

Only the parameters in low-rank matrices A and B called *adapters* are trainable. The effective new weights are made up by adding the frozen weights of the base model W_0 and ΔW, a dot product of the adapter matrices. Seen equation 6.2.2.1. Figure 6.13 explains the concept visually. LoRA can be applied selectively to any model layer. For transformers, it is typically applied to the query (Q) and value (V) projection matrices in the multi-head attention layers.

QLoRA (Quantized LoRA) [108] quantizes the frozen base model weights, in this case, to 4 bits, further reducing the memory footprint.

In practice, ΔW is scaled by a rank-dependent factor before being added to the base model weights. RsLoRA (rank-stabilized LoRA) [109] makes this scale inversely proportional to the $\sqrt{r}$, stabilizing LoRA fine-tuning.

Throughout the Llama 3.2 fine-tuning experiments, I used rank setting $r = 16$, which balanced computational requirements and learnable parameter size well.

Below, I will showcase a few select fine-tuning experiments on the Llama 3.2 model family.

Llama 3.2 models optionally come with instruction-tuned versions. These have been fine-tuned to follow text instructions. I found that even non-tuned versions benefit from additional context in the input. I generally used the following template for fine-tuning and inference (except for the responsive dataset):

```
Your job is to take an SVG file of a web design and convert it into a
pixel-perfect HTML and CSS markup and stylesheet.

### Input:
[SVG]

### Response:
[MARKUP]
```

Synthetic "Combined" dataset, 3B size When moving to the Llama 3.2 family from T5, the experiments on the synthetic data had to be carefully repeated. Llama 3.2 was expected to perform at least as well as T5, but this had to be verified.

For instance, in this experiment, I trained a 3B variant on the synthetic "Combined" dataset, which showed good performance across all synthetic datasets. Figure 6.14 shows the evolution of accuracy during training.

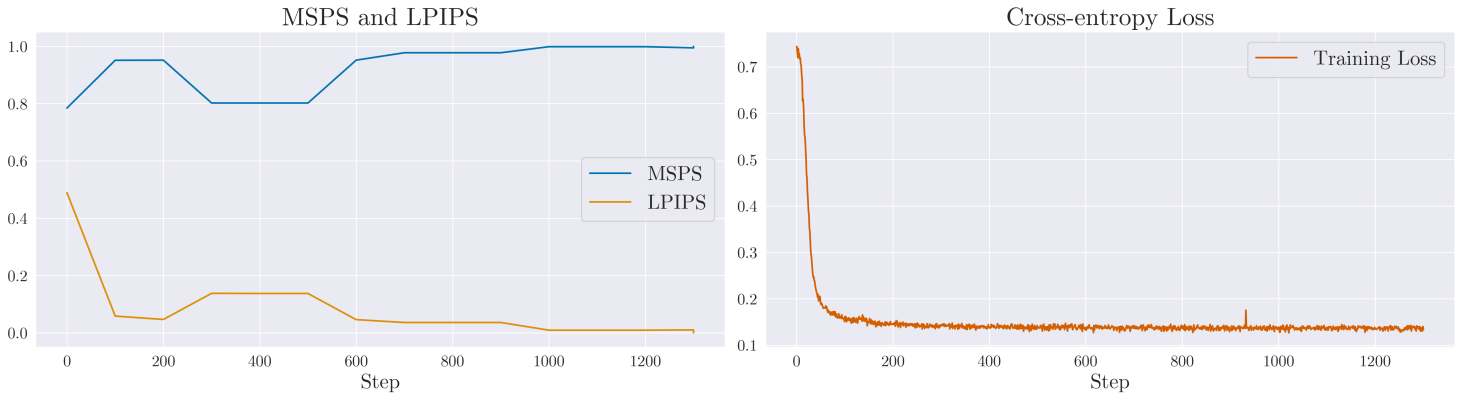


Figure 6.14: Llama 3.2 3B fine-tuning on the "Combined" synthetic dataset. The left-side chart shows the evolution of the accuracy metrics on the validation dataset as defined in 5. The right-side chart shows the cross-entropy loss on the training dataset. Due to an unsolved bug in Unsloth, the validation cross-entropy was not logged.

The total number of trainable parameters for this model and rank setting was $\approx$ 24 million.

The training took 7 hours 14 minutes or 5.68×10^{18} FLOPs for 1.40 epochs[8] The final test set scores are: **0.9947** MSPS and **0.0100** LPIPS.

With Llama's performance on the synthetic datasets verified, I moved on to public web datasets.

[8]For practical reasons, training length was defined in steps and not epochs for the Llama models. This resulted in fractional epochs in some cases.

Public "Chopped" dataset, 3B size In this experiment, I trained the 3B Llama 3.2 variant on the public web's "Chopped" dataset. The results (Figure 6.15) were poor. It appears that the model struggled to learn, and the accuracy metrics oscillate around the loss value from the base model without fine-tuning at step 0.

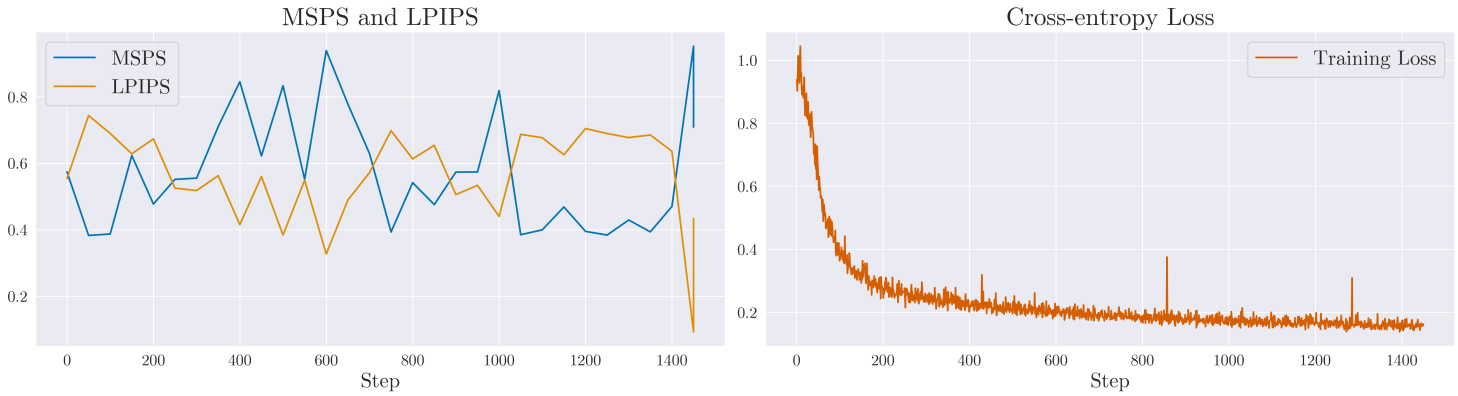


Figure 6.15: Llama 3.2 3B fine-tuning on the "Chopped" dataset. See the caption of Figure 6.14 for more information.

Public "Large" dataset, 3B size Similarly to the T5 family, I wanted to discard the possibility that the poor training performance was due to the idiosyncrasies of the "Chopped" dataset. In this experiment, I trained the 3B Llama 3.2 variant on the "Large" dataset. I considered this another failed experiment. It was stopped early after about 90 hours (0.24 epochs) because the cross-entropy loss flattened out and the accuracy metrics were not improving (Figure 6.16).

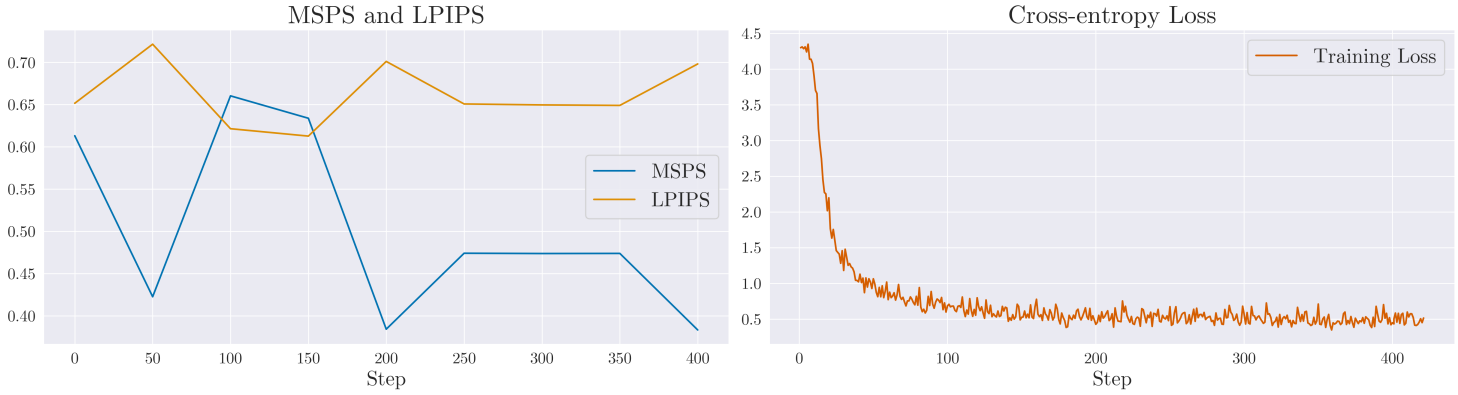


Figure 6.16: Llama 3.2 3B fine-tuning on the "Large" dataset. See the caption of Figure 6.14 for more information.

This made me think that perhaps the model was not large enough to learn the complexities present in the public web dataset. A model with 3 billion parameters is not trivial, especially compared to the T5 family, but with LoRA, we are only training a small fraction of those parameters.

Public "Large" dataset, 90B size I decided to go all-in on the size. The 90B variant is the largest in the Llama 3.2 family, and with the same rank setting, it has approximately 10 times as many trainable parameters as the 3B variant, approximately 265 million. Training this model on a large data set can become expensive. A single, one-epoch training on a length-filtered version of the "Large" dataset took over 392 hours (3.64×10^{20} FLOPs) on a single H100 GPU. For context, the cloud rental cost of this GPU was approximately US$2.70 per hour, so this was an experiment I could not afford to repeat many times. I wrote more on costs in section A.1.1.

Even with LoRA, the model's size reached the limits of single-GPU fine-tuning. The training dataset had to be filtered to contain only items shorter than 12,000 tokens. This limit was empirically determined by the out-of-memory errors I received during the training. The batch size was dropped to a single item with 128-step gradient accumulation.

The 90B variant of Llama 3.2 is a multi-modal model with vision capabilities. Since our dataset had only text modality, I explicitly excluded the vision layers from fine-tuning. It is unclear whether the vision layers contributed anything to the final results. These layers are built to work with bitmaps and convolutional layers. Still, higher-level, shared layers might have learned visual representations that indirectly contribute to the text-to-text performance.

Figure 6.17 shows mixed results.

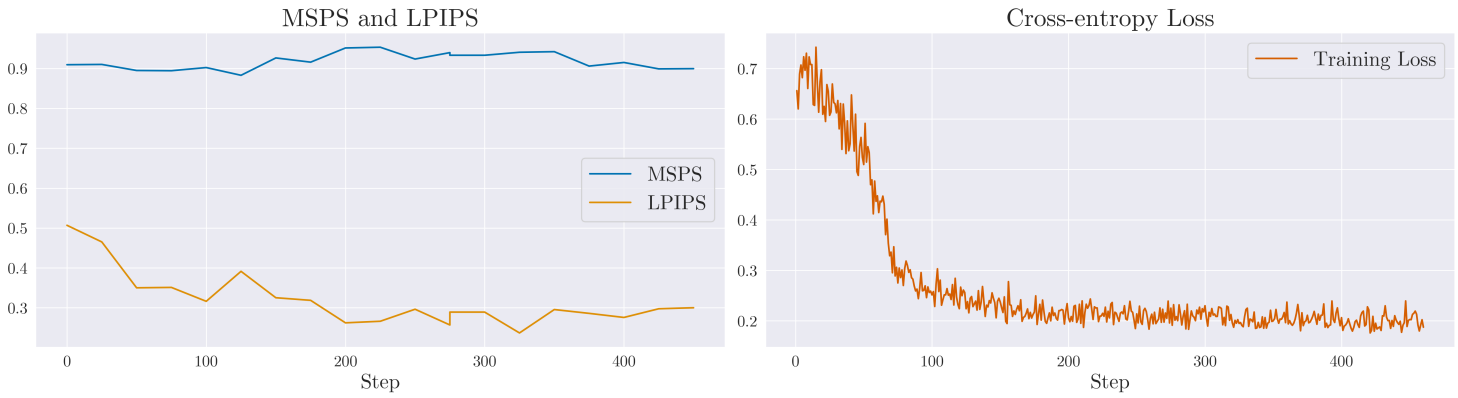


Figure 6.17: Llama 3.2 90B fine-tuning on the "Large" dataset. See the caption of Figure 6.14 for more information.

The accuracy metrics evolve more smoothly throughout the training. At step 0, the accuracy was already reasonably good. This is likely because the base model is much more capable and has some pre-existing understanding of the task. There was noticeable overfitting from the fourth quarter of the epoch. The best model was saved at 325 steps (MSPS: 0.9532, LPIPS: 0.2369).

The final test set scores are: **0.9530** MSPS and **0.3012** LPIPS. These scores appear to be suboptimal. TR3 For the model to be useful in a real-world setting, I estimate a

threshold of a minimum of 0.9900 MSPS and a maximum of 0.0100 LPIPS. More discussion on these results will follow in Chapter 7.

Still, this was the first experiment where the model demonstrated steady learning capability on the "Large" dataset. This indicates that it is able to extract useful correlations from a very complex, real-world data set and improve its performance. While this does not prove that a model that is valuable for real-world applications is certainly trainable, it hints that it might be possible and further investigation is needed. TR2

Perhaps higher-quality and larger training data sets, larger base models, more trainable parameters, and new training techniques can cross the threshold from research to industry.

At this point, continuing to train large models has become prohibitively expensive.

Unfortunately, the project's time, scope, and financial constraints were met before I could train a model of this size on several design resolutions to output responsive web pages. Thus, while the concept of responsive web pages from multiple input designs was proven in principle using very simple synthetic data, it remains to be determined whether it works on complex, public web datasets.

I published the 90B fine-tuned model under the Apache 2.0 license at:
https://huggingface.co/tcz/rb-llama-90b TR7

7 Results

In the previous chapter, I trained models on real-world websites to teach them to translate vector images to corresponding HTML and CSS code. The published star model, the 90B fine-tuned Llama 3.2, did not demonstrate very strong accuracy scores on the test set (**0.9530** MSPS and **0.3012** LPIPS). TR3 It is unlikely to be ready for industry use to aid software engineering or prototyping.

Nevertheless, as this model is the project's main output, I performed more testing beyond interpreting averaged accuracy scores on a test dataset.

7.0.1 Benchmark

No publicly available benchmark exists for models turning vector files into web pages.

However, there are two public benchmarks for bitmap-to-webpage tasks: Image2Struct [1] and WebCode2M [2]. Both test an array of multimodal LLMs, including some of the most powerful commercial models, on datasets the authors compiled. The tested LLMs are not fine-tuned on domain-specific data. Only Image2struct publishes a regularly updated leaderboard[1].

I want to be very explicit here by stating I did **not** benchmark my model with Image2Struct. It measures a different task, with a different difficulty, and distinct input and output formats. Still, testing my model on Image2Struct's web page dataset can be insightful. Thus, I ran their web page dataset through the crawler introduced in Chapter 4 to obtain SVG inputs for each web page. I then performed inference with

[1] https://crfm.stanford.edu/helm/image2struct/latest/#/leaderboard/image2webpage

them on my final model and calculated the LPIPS[2] and EMS (discussed in Chapter 4) scores for the generated HTML+CSS code.

Image2Struct leaderboards prefer the latter metric, although they publish LPIPS scores too. Table 7.1 shows the results.

Model	EMS	**EMS (vector)**	LPIPS*	**LPIPS* (vector)**	**Compilation success**	**Training and inference success**
GPT-4o (2024-08-06)	**0.735**		0.018		0.998	
Claude 3.5 Sonnet (20240620)	0.724		0.285		0.972	
Claude 3.5 Sonnet (20241022)	0.712		0.011		0.989	
GPT-4o (2024-05-13)	0.71		0.333		0.98	
GPT-4o mini (2024-07-18)	0.696		0.012		0.986	
Gemini 1.5 Pro (0409 preview)	0.683		0.367		0.971	
Gemini 1.5 Pro (002)	0.683		0.014		0.919	
Gemini 1.5 Flash (002)	0.674		0.008		0.916	
GPT-4V (1106 preview)	0.653		0.378		0.957	
Claude 3 Sonnet (20240229)	0.642		**0.383**		0.956	
Mistral Pixtral (2409)	0.642		0.015		0.921	
Palmyra Vision 003	0.64		0.311		0.884	
Gemini 1.0 Pro Vision	0.502		0.316		0.795	
LLaVA 1.6 (13B)	0.447		0.332		0.731	
LLaVA 1.5 (13B)	0.435		0.379		0.773	
Claude 3 Opus (20240229)	0.378		0.319		0.655	
Qwen-VL Chat	0.008		0.017		0.031	
IDEFICS-instruct (80B)	0.001		0.001		0.002	
IDEFICS-instruct (9B)	0.001		0.001		0.001	
IDEFICS 2 (8B)	0		0		0	
Reverse Browser (proposed)		0.693		0.755		0.981

Table 7.1: The performance of the final Reverse Browser model on a vector-to-webpage task, contrasted with multimodal LLM performance on bitmap-to-webpage as defined by Image2Struct. Both tasks use the same set of web pages for ground truth, but the scores are not comparable because they measure the performance on different tasks. The two different success rates are related to different failure modes, hence the separation. The updated LLM leaderboard was copied from the Image2Struct website on the 6th of May 2025.

* Image2Struct uses a normalized LPIPS where greater means better. They also use a neural network different from the one I used throughout this project (VGG over AlexNet). In this table, I adopt their metric.

Figure 7.1 shows a few examples of the model's output. In Appendix A.4, I write briefly on my methods of turning the web pages in the Image2Struct dataset into vector images and how inference was performed.

[2]Image2Struct authors normalize LPIPS to be a greater-is-better metric by subtracting the traditional LPIPS value from 1. They also use a different neural network to calculate LPIPS. They used VGG, I used AlexNet. Their paper was published on the 29th of October 2024, too late for me to adjust the LPIPS I used to match theirs.

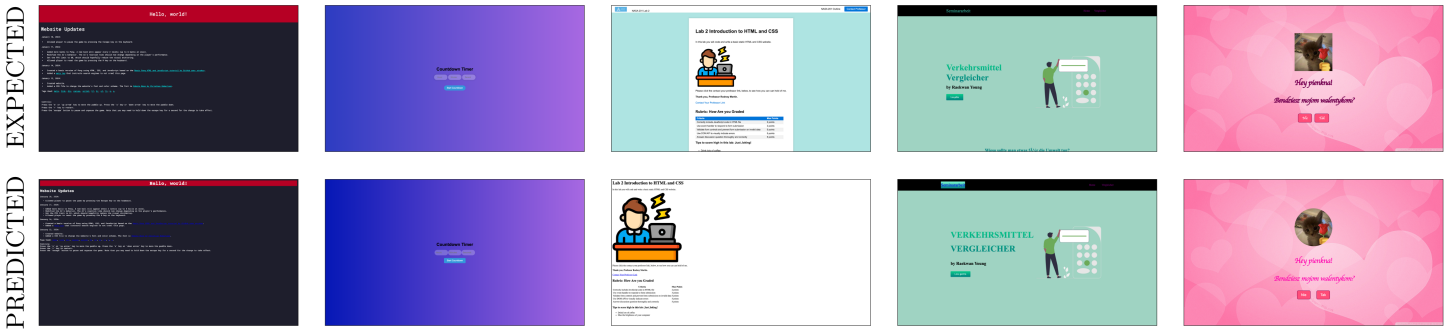


Figure 7.1: A few randomly selected examples from the model's output on the Image2Struct web pages dataset.

If comparing scores is not a fair game, what can we learn from these results? I have two main takeaways:

1. Even large, commercial, multimodal LLMs perform poorly at image-to-UI tasks. The Image2Struct prompts contain additional input besides the bitmaps (extracted text, list of resources), yet the results are not usable in a real software engineering setting. More research and development are needed before the industry can adopt image-to-UI.

2. A relatively small model trained by an MSc student with a limited budget can deliver accuracy scores similar (or, in the case of LPIPS, significantly better) to the best bitmap-to-webpage scores. This hints at vector images being a better choice for image-to-UI than bitmaps.

7.0.2 A work scenario

In this section, I perform more testing on the finished model and discuss techniques to improve its performance.

Let us assume I am a software engineer building web applications. I want to develop a simple, new web application. I start by coming up with a basic design for the website. I use the prototype drawing software Sketch[3] to progressively create the application's design. At this point, there's no coding involved; I'm placing and moving elements on the screen with the mouse.

Figure 7.2 shows the imaginary steps of creating the design. Each step adds an element or changes something small.

[3]https://www.sketch.com/

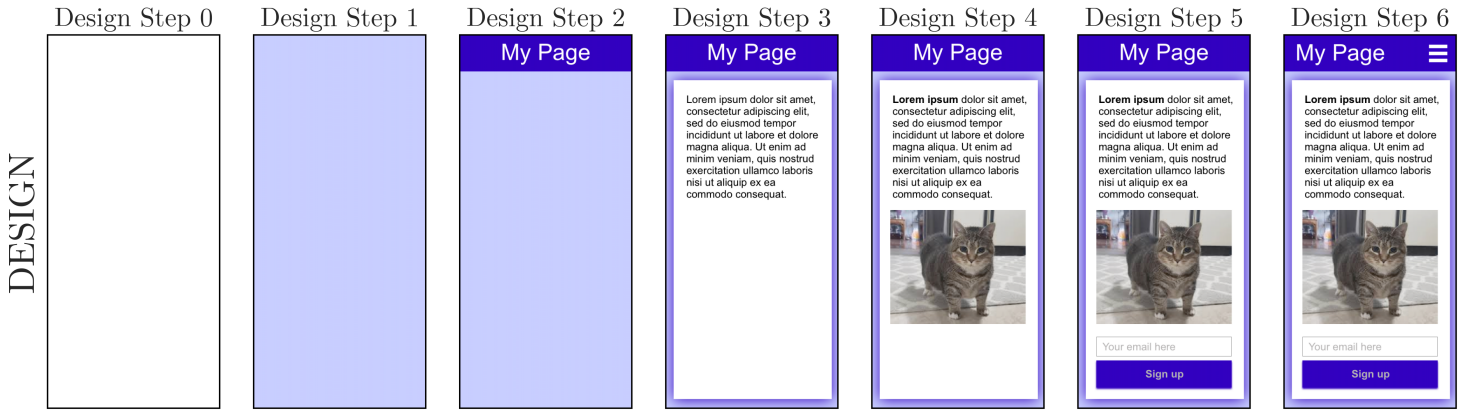


Figure 7.2: Evolution of a fictitious web design

With a vector-to-code tool like *Reverse Browser*, I may turn my design into HTML+CSS code and test it in a browser at each step. Once the design is complete, I can add dynamic features to the markup code. Turning even a simple design into code normally would require hours of complex software engineering work. A functional model may automate this task, taking mere seconds and the push of a button to generate HTML+CSS code.

Let us inspect how well the current model can turn our design into markup at each step.

To test this, I exported the designs at each step from Sketch and fed them into the trained model I prepared for inference. Then, I rendered the resulting markup in a browser and compared it to the bitmap version of the original. Notably, the SVG files from Sketch may have a very different "flavor" from the ones we created for the training datasets. Sketch may represent elements differently, use other units or grouping, or have dissimilar naming, so this is a vital test of the real-world utility of our model. Appendix A.5 contains a sample SVG file from this section.

The results seen on Figure 7.3 are poor.

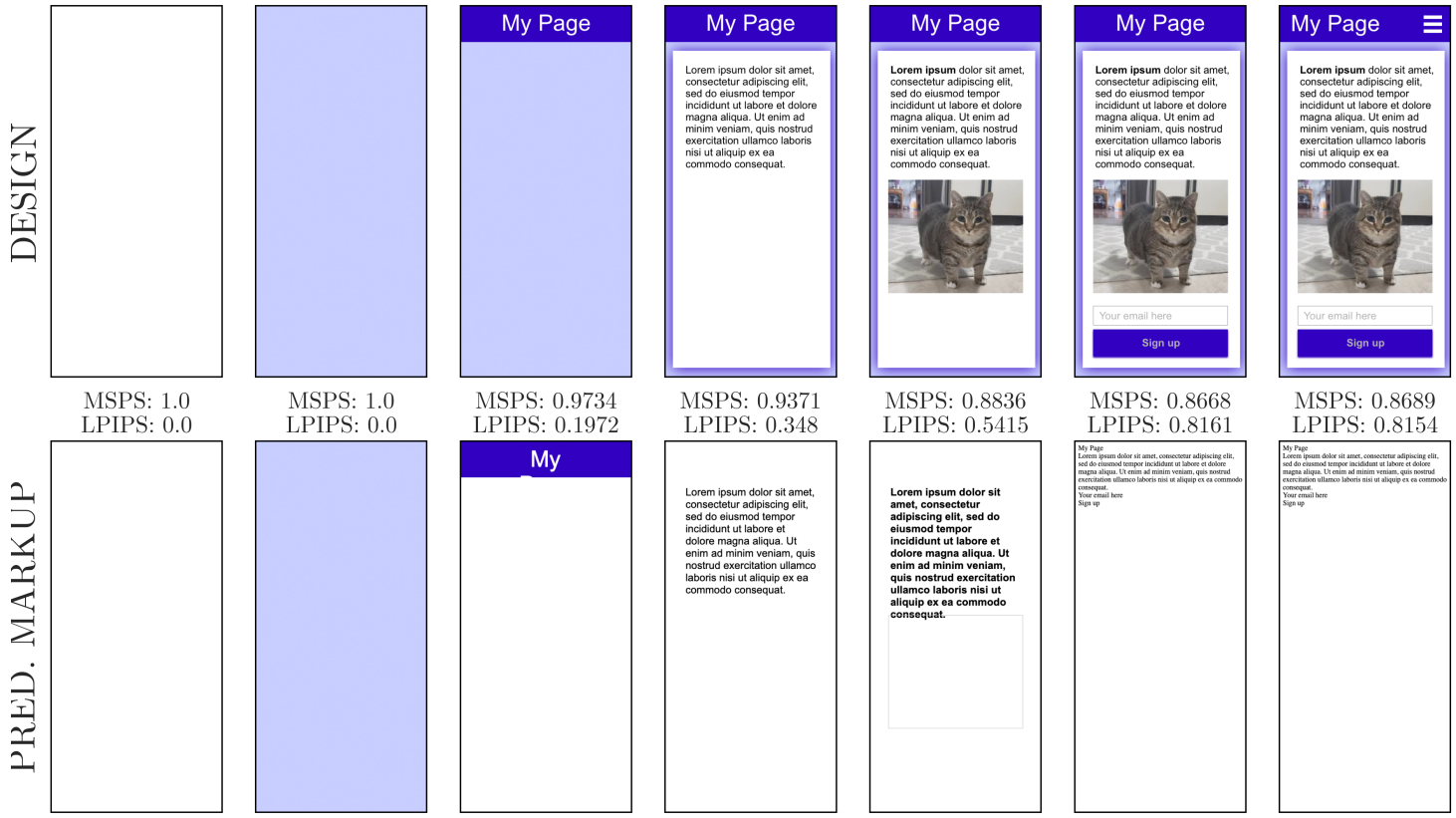


Figure 7.3: Model output for each design step with accuracy metrics between the original design and the resulting web page screen capture.

The model matched the website's background at step 2 and made a flawed header at step 3, but not much else. At this point, it may be questioned whether the model has any utility.

But there is a trick up our sleeve.

In this task, we always know the ground truth, and we can use it to precisely and automatically rate the output of the model. The accuracy of a generated web page can be compared with the design, and if the result is poor, we may try again. This is a well-known technique called best-of-N sampling [45], which trades test-time compute for better output quality. It is conditioned on having a scoring mechanism to choose the best response. Best-of-N sampling is often used with models trained to solve math or programming problems where the solutions are automatically verifiable for correctness. TR3

Let us see how the model performs if it has several shots to get the design right. The web pages on Figure 7.4 were picked from 25 samples for each design step, based on their LPIPS loss. The generation temperature was increased to 0.8 (from 0.0 previously), and top-p was set to 0.95. Top-p is a generation parameter and determines the set of token candidates to consider when generating the next token. The 0.95 setting means that 95% of candidates, weighted by the probability mass, will be considered. The temperature setting scales the differences in the probability distribution between the token candidates (logits).

A lower temperature magnifies minor differences, resulting in less stochastic output, while a higher temperature evens them out, resulting in more stochastic (or "creative") output.

Figure 7.4: The best model output by LPIPS out of 25 samples for each design step, with accuracy metrics between the original design and the resulting web page screen capture.

Best-of-N sampling and a higher temperature gave us much more accurate results. To appreciate the power of best-of-N sampling, let us look at the *worst-performing* output from the same generation run for each design step in Figure 7.5.

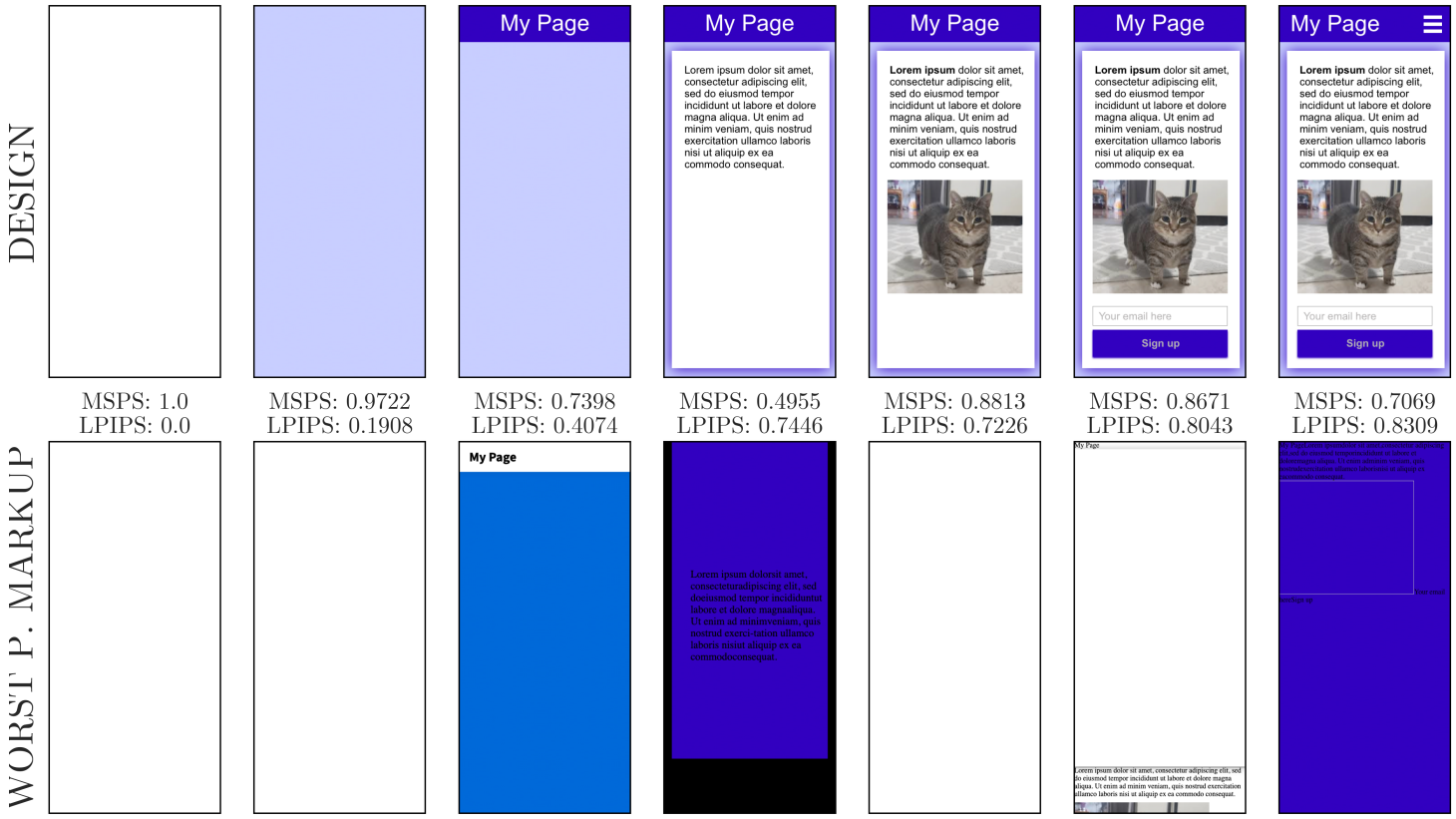


Figure 7.5: The worst model output by LPIPS out of 25 samples for each design step, with accuracy metrics between the original design and the resulting web page screen capture.

Are we unduly optimistic here? Are the results too good to be true? They certainly are. Examining the source code of the best-performing web pages reveals that the model has learned to "cheat." Since SVG files can be embedded on a webpage and a browser engine can render them, the model sometimes just took the input SVG, inserted it between `<body>` tags, and repeated it nearly verbatim. Naturally, these instances performed exceptionally well, but this is not what we want when we expect a coded web page.

When we repeat the scoring but exclude pages where a single embedded SVG takes up more than 50% of the viewport area, the results are much less exciting (Figure 7.6).

Figure 7.6: The best model output by LPIPS out of 25 samples for each design step, with accuracy metrics between the original design and the resulting web page screen capture. Here, candidates with large embedded SVG elements were excluded.

The filter discarded 6 out of 25 candidates for the most complex design step. Generating these to begin with is wasteful. A different strategy forbids the model from generating any embedded SVGs. Figure 7.7 shows the results when adding `svg` to the "bad words" configuration of the model's sampling parameters.

Figure 7.7: The best model output by LPIPS out of 25 samples for each design step, with accuracy metrics between the original design and the resulting web page screen capture. Here, generating `svg` elements was forbidden.

Can we trade more test-time computing for even better results? Indeed, although it is likely that the returns would eventually diminish. See Figure 7.8 for another example (samples: 50, temperature: 0.5, top-p: 0.9). Given time and money constraints, the right balance of compute resources, sample size, and temperature setting can be empirically determined. The current model is relatively slow and expensive to run.

Figure 7.8: The best model output by LPIPS out of 50 samples for each design step, with accuracy metrics between the original design and the resulting web page screen capture. Here, generating `svg` elements was forbidden, and a lower temperature of 0.5 was used.

7.0.3 Resource use

For inference, the adapter-merged model was quantized to `bfloat16` parameters. It was then loaded in vLLM [73], a high-throughput large language model inference engine. Due to the size of the final model, a minimum of four H100 GPUs (96GB of VRAM each) were needed to store model weights and the corresponding KV cache. The on-demand machine rental with this configuration was expensive, about US$9.50 per hour. TR6

The time to generate a web page design varies significantly by the complexity of the input SVG. For the design examples above, the time ranged from one second to ≈ 150 seconds. The per-token speed ranges from ≈ 1000 to ≈ 30 tokens per second for ingestion and ≈ 150 to ≈ 60 tokens per second for generation.

The high cost and slow speed cast further doubt on the model's usefulness, especially when used with best-of-N sampling. Still, if a hypothetical model were to output high-accuracy, high code quality results, the time and financial costs could be contrasted against performing the same task manually. In other words, if the model were accurate, the resource use would be economically reasonable even at this rate. Unfortunately, as seen above, the model's performance is far from highly accurate, even for simple designs.

Only a small fraction of the model's weights contribute meaningfully to the generated designs. The fact that Rome is Italy's capital is certainly not essential knowledge for generating web page code from vector images. Parameter pruning [110] is a common optimization technique that discards model weights (or entire layers) that don't meaningfully contribute to the performance and thereby reduces the overall model size. Pruning the trained model is likely to reduce the resource requirements massively. Due to the project's scope, I did not prune the finished model.

Emphasis on the code quality of the frontend code output by the model was explicitly out of scope for this project. So here I only offer a short, subjective evaluation: it is not very good.

8 Conclusions

In summary, the project concluded with mixed results. Most of the lower-level goals have been successfully achieved, but the crowning achievement was missing. PG

Data I consider this part a success.

High-quality and high-cardinality datasets were produced and published, including three public web datasets that took serious effort to create. DG1 DG2 All functional requirements were met, although I made a few questionable decisions that I would revisit if I started again. For instance, I would treat external web resources (images, perhaps fonts) differently. While I still think it is impractical to store external resources for a dataset of hundreds of thousands of websites, I would retain image metadata, and at test-time (both for the validation and test set) serve mock files with the same dimensions and formats as the original. I would also exclude "Not Found" pages from the dataset.

Although unsure of the specifics, I would attempt to simplify data instances. I suspect that overly complex and lengthy items hurt the final model quality and considerably slowed down the training. The "Chopped" dataset was meant to achieve this goal, but it fell short in practice. I would also try to filter for "visual interestingness" in some way because I noticed a high number of plain training examples that are unlikely to have contributed.

Looking back, I would also increase the size of the "Small Validation" to contain at least a dozen items to avoid wild swings in validation loss.

Metrics I am content with the results here.

The chosen metrics performed reasonably well. MG1 I was happy to find that LPIPS was also chosen by Roberts et al. [1], validating my choice. In hindsight, I am less certain that introducing MSPS was necessary. The reasoning still makes sense on

paper; the implementation did not take too much effort, and it was a valuable exercise. However, in practice, MSPS and LPIPS were moderately strongly correlated[1]. The threshold to filter the training dataset proved reasonable, but I would have liked to get it validated by human observers if I had had time. MG2 The scale of MSPS is a little counterintuitive. In retrospect, I would make it a lower-is-better metric and would use a scaling exponent for a more natural scale, so that $\text{MSPS}_{\text{new}} = 1 - (\text{MSPS}_{\text{old}})^{\alpha}$ where α is a small positive scaler constant.

All functional requirements were met.

Still, I think no current FR-IQA metric is excellent at capturing the nuances of this domain. IQA treats all pixels equally, but on user interfaces, not all pixels are created equal. For instance, with the current metrics, one could achieve high accuracy scores by getting the background color right while completely messing up the placement or look of a button or an important label. Roberts et al. [1] grapple with the same problem, but, in my opinion, their proposed new metric (EMD) falls short.

Models This part did not meet expectations.

I am happy with the choice of model architectures. They performed in line with the parameters based on which I chose them. TG1 I considered training on synthetic data successful, recognizing that the training data was simplistic and homogeneous, very far in complexity from a real-world dataset. Still, those experiments aimed to prove the isolated, specific "skills" of the trained models before moving on to more complex data. In that, they served their purpose. TG2 Before training the final model, I tried to remove uncertainties: I estimated the required data and model sizes and gathered information from related work. Even so, the outcome had always had a high degree of uncertainty due to the nature of large-scale machine learning projects. The final results were only available when the model training was done.

At the end of the project, I reached the limit of the allotted timeframe and financial budget, so I could not continue with high-scale training experiments. The final model does not meet the goals. TG3 In particular, it falls short on accuracy TR3 and is unable to produce responsive web pages from different design resolutions TR2 (although on the latter, I completed successful PoC experiments on synthetic data). Yet, the final model has demonstrated a significant leap in accuracy compared to the base model, proving that the concept has legs. Its accuracy fell in line with benchmarked LLMs tested on a bitmap-to-webpage task.

[1]Pearson correlation of ≈ -0.62 with p-value of 1.19×10^{-95} on the Image2struct sample

The big picture is bright.

I did not fully achieve the project's principal goal, but I accept that it carried a high level of risk. I, however, feel happy that I have achieved a great learning outcome. I acquired deep knowledge on state-of-the-art AI models and training techniques, IQA metrics, and scaling web crawling. I gained hundreds of hours of hands-on experience in these areas. While writing this report, I also learned considerable skills in using TeX. I hope others will find the published datasets, model, and code useful.

While I did not succeed in creating a model ready for prime time, I am more convinced than ever that it is possible. With more resources and a more competent team of researchers, someone is bound to make it happen. I intend to continue working in this domain, trying new techniques such as Reinforcement Learning with Verifiable Rewards [111], training new models, such as the recently released Llama 4[2], and attempting to correct my mistakes in this project.

Financials and elitism Having completed this experience, I became deeply concerned about the financial costs of getting hands-on experience with modern machine learning and AI techniques. As detailed in section A.1.1, I did not get external funding for this project, and I spent a considerable amount of my funds on computing resources. I am fortunate enough to have been able to do that. I am in a small minority.

I grew up in an era when computing was somewhat democratic. I was born in a small, relatively poor post-communist country and raised by a single mother. Still, we could afford a computer, albeit a little obsolete, yet capable of teaching me just about anything I wanted to learn about computing. This is no longer the case in a world with billion-parameter models, terabytes of data and GPUs that cost US$30,000 each.

In this world, students and tech-savvy kids risk falling behind, and important technologies (perhaps the most important ones in 2025) become gatekept by the elite. I do not have a solution. I believe that investment by schools, universities, and governments in affordable (or free) computing infrastructure could be a good step forward.

[2]https://ai.meta.com/blog/llama-4-multimodal-intelligence/

Appendices

A
Appendix

A.1 Project management

Different parts of the project required different project management strategies.

For the less risky sub-goals where the methods and tasks were well-defined from the beginning, and the outcome was not highly uncertain, a simple Kanban-style [112] flow worked well. These were the *Data* and *Metrics* parts. Since they were co-dependent, I worked on them in parallel. Initially, I thought carefully about the requirements, did some preliminary research, and defined small, maximum 1-2 days' worth of independent tasks. I used a note-taking app called Notion[1] to visualize the work (pending, in-progress, done) and limited my work-in-progress to one or two active items. Since I worked alone, I could afford a low level of formalities and follow my preferred prioritization technique: the next task to work on became the one I was most excited about at the time.

The riskier, more uncertain sub-goals related to model training and evaluation required a different cadence. Due to my inexperience in this area but also due to the nature of model training and tuning, the work here required a much more iterative and adaptive approach. Much of the work was about setting up a long training experiment, pressing "Run" and waiting hours, sometimes days, to see the outcome, then repeating. The *Model* part of the project permitted a lot of "play time", exploration of new concepts by trying and tweaking. I enjoyed this part immensely. What I did not enjoy was the frequent encounter with bugs, instability, and the lack of documentation, which I wrote about in Chapter 6.

[1] https://www.notion.com/

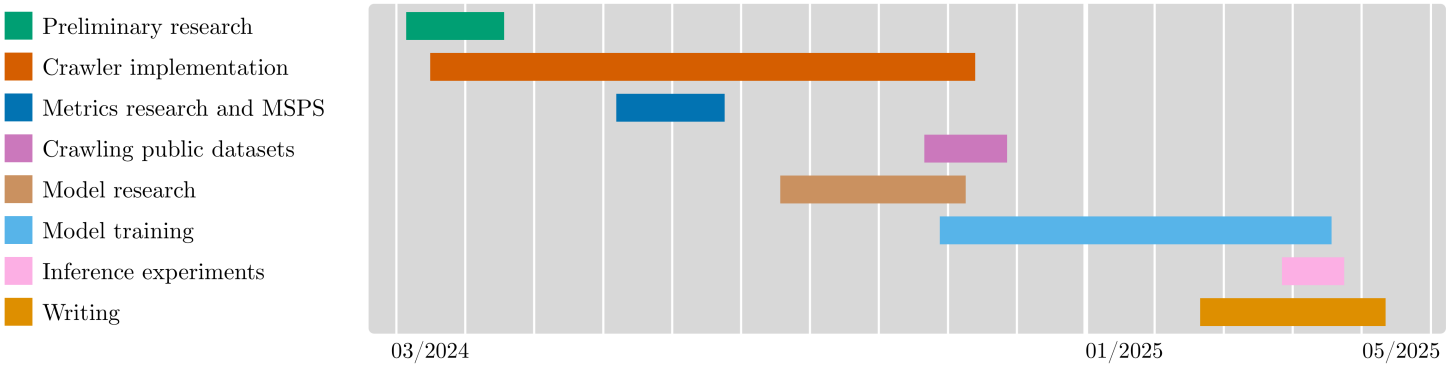


Figure A.1: Gantt chart of the project timeline

I wrote about risks associated with each subgoal in Chapter 3. I tried to manage that risk by collecting information upfront and working out proofs of concept, as described in the relevant chapters. I also planned large margins for error accordingly, both in time and financial resources. The project stretched the limits of both of those and ended up taking much longer and costing much more money than I hoped for. Figure A.1 contains a rough project timeline chart. Progress was necessarily part-time due to simultaneous personal and professional commitments.

Throughout the project, I often used the Pomodoro [113] technique to help me focus.

A.1.1 Financial costs

From the beginning of the project, it was clear that GPU training would require non-trivial funding. I do not own a high-end GPU fit for machine learning, and I expected that GPU training times would be in the hundreds of hours, maybe more.

Additionally, training state-of-the-art models requires state-of-the-art GPUs. This requirement was further demanded by the long input sequence length of the training dataset, which was also predictable at the outset. For instance, the Llama 3.2 models from 11B parameters and above cannot be fine-tuned on a GPU less powerful than an Nvidia H100. Even on that hardware, many optimization tricks were needed (see Chapter 6) for fine-tuning.

At the beginning of the project, I attempted to obtain funding from Amazon's AWS Cloud Credit for Research[2]. I never heard back from them. There are a few other GPU grant programs (Google, Microsoft, Oracle) that were not designed for MSc students.

The Department of Computer Science of the University of Oxford, through the advocacy of my supervisor, could have offered some computing credits at Oxford's Advanced Research Computing (ARC), but at the time, they did not seem to have any H100 GPUs available (they do now) and it was not clear how many credits I would get and under what schedule.

[2]https://aws.amazon.com/government-education/research-and-technical-computing/cloud-credit-for-research/

Finally, due to the unknowns associated with the model training part of the project (largely as a result of my own inexperience), I opted for self-funding.

This was a considerably more comfortable situation and allowed much-needed free exploration, but it came at a high cost. The project's total cost ended up at approximately **£7,400**. Appendix A.7 contains a detailed breakdown. In Chapter 8, I reflected briefly on the financial aspects of this area of research.

A.2 Ethics and social considerations

The 20th-century German philosopher, Martin Heidegger, probably would not like contemporary Artificial Intelligence.

Certainly not the kind that enjoys mass popularity in 2025. "Everywhere everything is ordered to stand by, to be immediately on hand, indeed to stand there just so that it may be on call for a further ordering", he wrote in his 1954 work *The Question Concerning Technology* [114]. He meant that by applying modern technology to our natural surroundings, we reduce nature to a stockpile of resources. He illustrated the "monstrousness that reigns here" with the Rhine river converted into and reduced to a mere power source, and contrasted "'The Rhine,'as dammed up into the power works, and 'The Rhine,'as uttered by the art work, in Hölderlin's hymn by that name", something a lot more than just water molecules with potential energy to be captured and stored. He goes further and explains that humans themselves become mere resources and are forced to partake and keep modern technology ticking: "If man is challenged, ordered, to do this, then does not man himself belong even more originally than nature within the standing-reserve?"

I cannot help but see the parallel with modern AI. Everything that has ever been created, all books, articles, conversations, and video and audio recordings, is gobbled up by large, hungry models to extract their essence, store it, and resell it. They ask for no permission. Not just humanity's intellectual products but also people's activity (and so people themselves) are reduced to data points to feed an algorithm. The result is often the concentration of value and power in a few hands and the erosion of the free will of humans. Algorithms fed on human creativity and behavior tell us what to read, what to buy, and what to think.

My project seems harmless in contrast, but has similar characteristics: I took a non-trivial number of public web designs and attempted to extract their essence. The resulting model is very limited, and it is unlikely to be truly useful to anyone. Nevertheless, it is conceivable that with more data, better models, and more careful training, it can evolve to have higher utility in the future.

Heidegger points to the problem but does not offer a clear solution in this work. He also does not discuss the impact of modern technology on the job market.

In a 2020 World Economic Forum (WEF) report titled *The Future of Jobs Report* the authors estimate that "by 2025, 85 million jobs may be displaced by a shift in the division of labour between humans and machines, while 97 million new roles may emerge that are more adapted to the new division of labour between humans" [115]. Now, in the year the prediction is due, we see no strong evidence of those numbers unfolding. In a 2023 WEF report [116], the authors claim that "Software Developers also perform many tasks with high potential for automation, suggesting that many jobs will be transformed rather than automated or augmented."

It seems even experts are unsure how and when ever-more-powerful AI models will affect the job market.

Today, AI coding tools are gaining ground, making software developers more productive rather than replacing them. Nevertheless, it is perhaps naive to think that this will always be the case. Will a skilled human in the loop be necessary when a powerful system can take over 90% of the work? How about 99.9%? Will "software development" of the future be simply conversing with a computer using natural language and telling it how we want it to behave? If so, how many skilled persons will be needed who deeply understand the inner workings of the machine? I am skeptical that software developers have long job prospects. Ironically, the work of many of them directly contributes to making them obsolete.

There are more optimistic readings of AI.

Acclaimed computer scientist and book author Jaron Lanier refers to modern AI as "an innovative form of social collaboration" [117]. He advocates for transparency regarding how these models are built and "data dignity", a concept of fairly compensating data contributors.

I am inspired by the work of computer scientist and interface designer Bret Victor [118], who emphasizes that the right tools can unlock unprecedented levels of creativity. While he does not explicitly talk about AI, I believe that, if *Reverse Browser* were to materialize with the accuracy I envisioned, it could serve as such a tool, enabling software engineers to focus on the creative aspects of their work and avoid sinking effort into "plumbing" work.

Turning an existing web design into code is not a particularly creative or enjoyable task, in my opinion. Borrowing Robert Pirsig's expression, it is a "gumption trap." In *Zen and the Art of Motorcycle Maintenance*, he writes beautifully about what happens when those traps are removed between the craftsman and the project at hand: "The material and the craftsman's thoughts change together in a progression of smooth, even changes until his mind is at rest at the exact instant the material is right." [119]

For this project, I aspired to build a tool that empowers "craftsmen" (software engineers) rather than replacing them.

I tried my best to be respectful in the way I collected data. I write about my `robots.txt` strategy in Chapter 4. Still, the bot-exclusion mechanism is opt-out by design, and

implicitly allowing crawlers to access a website does not imply consent to data collection. It would also be unreasonable to expect website authors to predict new ways their data could be used and take proactive steps to prevent it, although evidently they started excluding AI crawlers en masse [120].

I published and open-sourced all output from this project under a permissive license in hopes that someone will find it useful and that it mitigates the very valid ethical concerns related to this project.

I cannot conclude the ethics section without mentioning the environmental impact of this project.

Computers, and especially GPUs, are power-hungry machines, so I directly and significantly contributed to greenhouse gas emissions through this project. Information on the exact power consumption or the energy mix is unavailable when renting through cloud providers. I relied on findings in a paper by Song, Chen and Zong [121] and calculated with $\approx 0.2kg$ of carbon emissions per GPU training hour, or 338kg for all the documented training hours.

I applied a 2× multiplier to account for non-documented training runs, inference, crawler machine time, energy spent setting up dependencies, etc. I hired a Direct Air Capture carbon removal company called Climeworks[3] to remove 677kg of CO_2 from the atmosphere. See Appendix A.6.

A.3 Legal considerations

The recent surge of generative AI technologies has sparked heated arguments about the legality of data collection, both in public discourse and courtrooms.

Generative AI companies such as OpenAI, Anthropic, Google, and Microsoft have built powerful AI models from massive amounts of data they collected predominantly from public sources. Beyond ethics, the practice raised serious legal questions and prompted various lawsuits: book authors and news publishers sued OpenAI, Meta, and others [122–124]. Developers sued OpenAI and GitHub [125]. Artists and stock photo services sued Stability AI and Midjourney [126, 127]. All of these suits are still pending.

AI companies generally argue that their data collection falls under fair use and that no direct harm has been caused to the original rightsholders. The other side argues that their copyright has been infringed, and generative AI models serve as substitutes for the original work.

Different legal frameworks in major jurisdictions further complicate the situation.

[3] https://climeworks.com/

United States The U.S. copyright law provides a critical exception for fair use (17 U.S.C. §107) [128]. The criteria boil down to whether the new use is transformative compared to the original and whether it affects the original work's market. Google successfully used this exception to defend its right to build a searchable database of books in the lawsuit Authors Guild, Inc. v. Google, Inc. [129].

In Chapter 4, I cited a paper by Jeon and Roy [23] that establishes the connection between model learning and lossy compression. From a software engineer's perspective, it seems the legal arguments hinge on how "lossy" these models are. Can they recite entire articles verbatim or mimic artistic styles unmistakably?

European Union The EU does not have a broad fair use exception. However, Directive 2019/790 Article 4 [130] provides an explicit exception for data mining that pushes the responsibility of opting out to the rightsholder (e.g., by the use of `robots.txt`). The new EU Artificial Intelligence Act (Directive 2024/1689) [131], which took effect in February 2025, prescribes obligations to the developers of "base models" such as OpenAI or Google concerning transparency on the source of data they used for model training. It also mandates respecting opt-out data collection and forbids some privacy-related scraping (e.g., CCTV cameras, massive sources containing facial information).

United Kingdom The UK has much stricter copyright laws that do not provide a fair use exception for commercial uses and permits data mining for non-commercial research only ("fair dealing") [132]. In fear of falling behind in AI development, the UK parliament issued a consultation [133] with the intent to redefine copyright regulation for AI training. It has been suggested to "avoid overly prescriptive provisions" and possibly adopt a model similar to the EU's.

In summary, the legal environment surrounding data scraping for AI models is uncertain and fluid in 2025. Importantly, all major jurisdictions provide an exception from copyright for research purposes, which gives this project a solid legal footing.

A.4 On the Image2Struct benchmark

In this section, for transparency, I will explain additional details about how the Image2Struct modified benchmark seen in section 7.0.1 was performed.

The Image2Struct [1] web page dataset[4] contains 900 web pages obtained from public GitHub Pages repositories. These are stored as individual ZIP files and include all the local assets (HTML, CSS, JavaScript, images) obtained from the repositories. The

[4]https://huggingface.co/datasets/stanford-crfm/image2struct-webpage-v1

dataset is partitioned into three equally sized splits: HTML, CSS, and JavaScript. In my correspondence with the authors, they confirmed what these splits mean. They indicate that one of the three languages is more heavily represented in a given web page repository.

Image2Struct leaderboards show the combined score using the entire dataset and also publish separate scores for each split. I combined the splits into one dataset and only calculated the overall score. This also meant that the dataset had many JavaScript-heavy pages. Reverse Browser's renderer explicitly disables JavaScript in the crawled pages so that dynamic features do not jeopardize the image capture (both SVG and bitmaps), and the model is not trained to generate JavaScript.

Image2Struct runs a local Jekyll[5] web server (also used by GitHub Pages) to serve the web pages and capture their content. I did the same, starting a separate server per page, and used the list of local URLs as input to my web page renderer described in section 4.1.

Image2Struct captures the screen at 1920×1080 pixel resolution and discards all content outside the viewport when taking a screen capture. This is unlike the behavior of my renderer (which captures the entire page), so I made changes to follow this method. When crawling, I removed all DOM elements outside the viewport and captured SVG with only the remaining elements. I also saved all image resources that the web page loaded. These were made available later when rendering the pages generated by the model.

I then created a dataset with the obtained SVG-markup pairs. This time, I did not discard any items based on their MSPS score, resulting in a dataset of 884 items. Of the original 900 web pages, 16 had severe syntactic errors that prevented the SVG conversion.

I then performed inference on the trained 90B Llama model using the SVGs and the same prompt I used during training. The notebook and the generated pages were published in the experiment repository[6]. Image2struct uses a 0.0 temperature setting to avoid fluctuations in the performance. I followed the same strategy. They also do not use best-of-N prompting, nor did I. They have no documented mechanisms to catch the models "cheating", e.g., simply presenting a web page containing the image from the prompt. I also did not use any bad-word filters (as discussed in section 7.0.2), but only a small percentage of the outputs contained text referring to SVG.

The final scores are based on 882 items as reflected in Table 7.1. An additional two items were "lost" due to web page rendering problems with the output. The table separates *Compilation success* as defined by Image2Struct and *Training and inference success* that are influenced by different error modes as described here.

[5] https://jekyllrb.com/

[6] https://github.com/tcz/rb-experiments/blob/main/helm-generations/vllm-image2struct-inference.ipynb

A.5 Design SVG sample

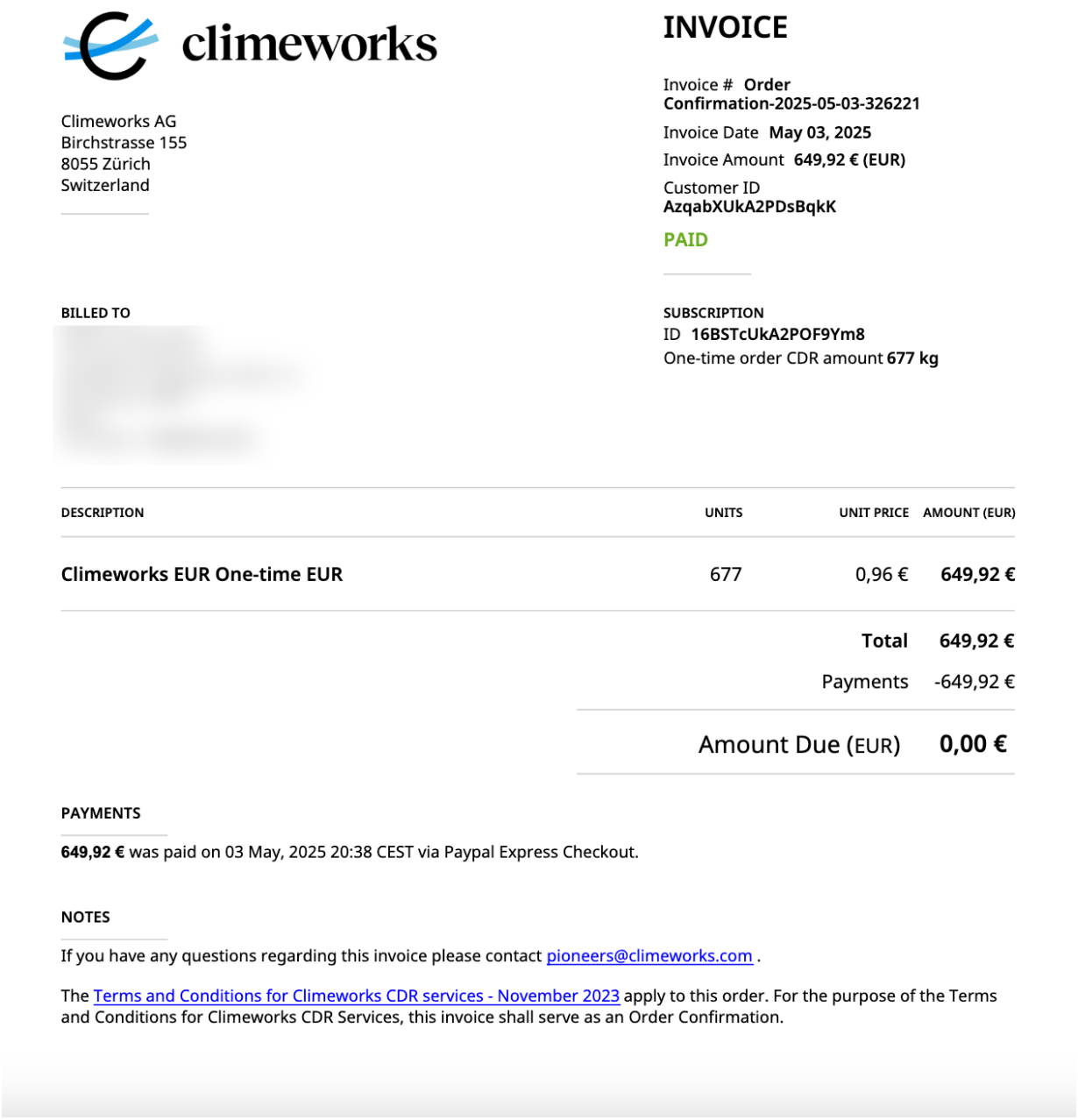
In Chapter 7, I refer to SVG files obtained from the graphics program Sketch as part of a mock design process. Below, I present an example SVG used for inference. Note that the image file, which would normally be embedded in the SVG file, was replaced with an external reference.

```
<?xml version="1.0" encoding="UTF-8"?>
<svg width="393px" height="852px" viewBox="0␣0␣393␣852" version="1.1" xmlns="
    http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
    <title>7</title>
    <defs>
        <rect id="path-1" x="18" y="104" width="361" height="727"></rect>
        <filter x="-14.4%" y="-6.9%" width="128.8%" height="114.3%" filterUnits=
            "objectBoundingBox" id="filter-2">
            <feOffset dx="0" dy="2" in="SourceAlpha" result="shadowOffsetOuter1">
                </feOffset>
            <feGaussianBlur stdDeviation="17" in="shadowOffsetOuter1" result="
                shadowBlurOuter1"></feGaussianBlur>
            <feColorMatrix values="0␣0␣0␣0␣0.196078431␣␣␣␣0␣0␣0␣0␣0␣␣␣␣0␣0␣0␣0␣
                0.752941176␣␣0␣0␣0␣1␣0" type="matrix" in="shadowBlurOuter1"></
                feColorMatrix>
        </filter>
        <filter x="-2.4%" y="-7.3%" width="104.9%" height="124.2%" filterUnits="
            objectBoundingBox" id="filter-3">
            <feOffset dx="0" dy="3" in="SourceAlpha" result="shadowOffsetOuter1">
                </feOffset>
            <feGaussianBlur stdDeviation="2" in="shadowOffsetOuter1" result="
                shadowBlurOuter1"></feGaussianBlur>
            <feColorMatrix values="0␣0␣0␣0␣0.196078431␣␣␣␣0␣0␣0␣0␣0␣␣␣␣0␣0␣0␣0␣
                0.752941176␣␣0␣0␣0␣1␣0" type="matrix" in="shadowBlurOuter1"
                result="shadowMatrixOuter1"></feColorMatrix>
            <feMerge>
                <feMergeNode in="shadowMatrixOuter1"></feMergeNode>
                <feMergeNode in="SourceGraphic"></feMergeNode>
            </feMerge>
        </filter>
    </defs>
    <g id="7" stroke="none" stroke-width="1" fill="none" fill-rule="evenodd">
        <rect fill="#C8CEFF" x="0" y="0" width="393" height="852"></rect>
        <rect id="Rectangle" fill="#3200C0" x="0" y="0" width="393" height="84"><
            /rect>
        <text id="My-Page" font-family="ArialMT,␣Arial" font-size="51.8" font-
            weight="normal" fill="#FFFFFF">
            <tspan x="26.7911133" y="59">My Page</tspan>
        </text>
        <g id="Rectangle">
            <use fill="black" fill-opacity="1" filter="url(#filter-2)" xlink:href
                ="#path-1"></use>
```

```
            <use fill="#FFFFFF" fill-rule="evenodd" xlink:href="#path-1"></use>
        </g>
        <text id="Lorem-ipsum-dolor-si" font-family="Arial-BoldMT,␣Arial" font-
            size="24" font-weight="bold" fill="#000000">
            <tspan x="47" y="156">Lorem ipsum</tspan>
            <tspan x="197.691406" y="156" font-family="ArialMT,␣Arial" font-
                weight="normal"> dolor sit amet, </tspan>
            <tspan x="47" y="183" font-family="ArialMT,␣Arial" font-weight="
                normal">consectetur adipiscing elit, </tspan>
            <tspan x="47" y="210" font-family="ArialMT,␣Arial" font-weight="
                normal">sed do eiusmod tempor </tspan>
            <tspan x="47" y="237" font-family="ArialMT,␣Arial" font-weight="
                normal">incididunt ut labore et dolore </tspan>
            <tspan x="47" y="264" font-family="ArialMT,␣Arial" font-weight="
                normal">magna aliqua. Ut enim ad </tspan>
            <tspan x="47" y="291" font-family="ArialMT,␣Arial" font-weight="
                normal">minim veniam, quis nostrud </tspan>
            <tspan x="47" y="318" font-family="ArialMT,␣Arial" font-weight="
                normal">exercitation ullamco laboris </tspan>
            <tspan x="47" y="345" font-family="ArialMT,␣Arial" font-weight="
                normal">nisi ut aliquip ex ea </tspan>
            <tspan x="47" y="372" font-family="ArialMT,␣Arial" font-weight="
                normal">commodo consequat.</tspan>
        </text>
        <image id="Bitmap" x="42" y="400" width="309.223301" height="260"
            xlink:href="cat.jpg"></image>
        <g id="Group" transform="translate(42,␣689)">
            <rect id="Rectangle" stroke="#979797" x="0.5" y="0.5" width="308"
                height="44"></rect>
            <text id="Your-email-here" font-family="ArialMT,␣Arial" font-size="24
                " font-weight="normal" fill="#B7B5B5">
                <tspan x="14" y="31">Your email here</tspan>
            </text>
        </g>
        <g id="Group-2" filter="url(#filter-3)" transform="translate(42,␣743)">
            <rect id="Rectangle" fill="#3200C0" x="0" y="0" width="309" height="
                62"></rect>
            <text id="Sign-up" font-family="Arial-BoldMT,␣Arial" font-size="24"
                font-weight="bold" fill="#B7B5B5">
                <tspan x="112" y="39">Sign up</tspan>
            </text>
        </g>
        <g id="Group-3" transform="translate(331,␣23)" fill="#FFFFFF">
            <rect id="Rectangle" x="0" y="0" width="42" height="8"></rect>
            <rect id="Rectangle" x="0" y="16" width="42" height="8"></rect>
            <rect id="Rectangle" x="0" y="32" width="42" height="8"></rect>
        </g>
    </g>
</svg>
```

A.6 Carbon capture receipt

As mentioned in section A.2, I attempted to offset the carbon emissions from this project. See Figure A.2 for the direct air capture receipt.



climeworks

Climeworks AG
Birchstrasse 155
8055 Zürich
Switzerland

INVOICE

Invoice # **Order Confirmation-2025-05-03-326221**
Invoice Date **May 03, 2025**
Invoice Amount **649,92 € (EUR)**
Customer ID **AzqabXUkA2PDsBqkK**
PAID

BILLED TO

SUBSCRIPTION
ID **16BSTcUkA2POF9Ym8**
One-time order CDR amount **677 kg**

DESCRIPTION	UNITS	UNIT PRICE	AMOUNT (EUR)
Climeworks EUR One-time EUR	677	0,96 €	**649,92 €**
		Total	**649,92 €**
		Payments	-649,92 €
		Amount Due (EUR)	**0,00 €**

PAYMENTS

649,92 € was paid on 03 May, 2025 20:38 CEST via Paypal Express Checkout.

NOTES

If you have any questions regarding this invoice please contact pioneers@climeworks.com .

The Terms and Conditions for Climeworks CDR services - November 2023 apply to this order. For the purpose of the Terms and Conditions for Climeworks CDR Services, this invoice shall serve as an Order Confirmation.

Figure A.2: Climeworks receipt

A.7 Detailed financials

Table A.1 contains the detailed breakdown of the financials associated with this project.

The lion's share of the costs was spent renting on-demand GPU instances from Vast.ai[7]. This was one of the most affordable cloud GPU rental services I could find. They operate a marketplace where third parties rent out their unused GPU instances. This comes with the trade-off that the instances do not have an indefinite lifetime and may be terminated at the owner's discretion. In all, I rented 2,733 hours of GPU time at an average rate of $2.81 per hour. These were almost exclusively Nvidia H100 instances, with the vast majority being single-GPU instances. I occasionally used multi-GPU instances for inference.

The second highest cost item was related to dedicated servers rented for crawling. Initially, I rented virtual machines (EC2) instances from Amazon Web Services, but these showed very poor value in terms of CPU performance per dollar during the web crawling. Therefore, I decided to rent dedicated servers from Hetzner[8], an affordable provider. These came with an initial setup cost, but worked out much cheaper and much more performant overall. I initially rented ten EX44 servers to crawl the public web datasets. Here, I had to balance time constraints with financial expenditure. Each of the three large public web datasets took approximately a week to complete on these machines. Each EX44 machine was configured with a 13th-generation Intel Core i5-13500 processor, 64 GB DDR4 RAM and two 512 GB Gen4 NVMe SSDs. They cost €42.30 per month with a one-time setup fee of €39.00. Later, I kept a single instance available for occasional smaller crawling jobs related to the synthetic datasets.

The third category was cloud storage on Amazon Web Services[9]. While not strictly necessary for the datasets, I decided to keep the screenshots of the web pages and SVGs for each item for the project's duration, along with their vector image file and markup. This turned out to be unnecessary, but I could not be sure if I wanted to perform additional analysis on them later, e.g., if I had to swap the IQA algorithm for some reason. The crawling activity of public web, synthetic, and development datasets resulted in an Amazon S3 bucket with 24,678,676 files occupying 4.2 terabytes.

[7] https://vast.ai/
[8] https://www.hetzner.com/
[9] https://aws.amazon.com/s3/

COST TYPE	MONTH	COST
Vast.ai GPU rental (USD)		
Download	2024-11	$0.01
	2025-05	$0.12
Download Total		$0.14
GPU	2024-09	$33.64
	2024-10	$222.02
	2024-11	$753.40
	2024-12	$1,509.81
	2025-01	$581.42
	2025-02	$938.46
	2025-03	$886.05
	2025-04	$1,715.36
	2025-05	$475.91
GPU Total		$7,116.08
Storage	2024-09	$4.87
	2024-10	$22.57
	2024-11	$7.92
	2024-12	$41.15
	2025-01	$16.28
	2025-02	$36.84
	2025-03	$130.18
	2025-04	$268.30
	2025-05	$26.85
Storage Total		$554.95
TOTAL		$7,671.17
Hetzner Dedicated Server for Crawling (EUR)		
EX44 Dedicated Server	2024-10	€147.60
	2024-11	€423.00
	2024-12	€46.17
	2025-01	€42.30
	2025-02	€42.30
	2025-03	€42.30
	2025-04	€42.30
EX44 Dedicated Server Total		€785.97
Primary IPv4	2024-10	€5.88
	2024-11	€17.00
	2024-12	€1.84
	2025-01	€1.70
	2025-02	€1.70
	2025-03	€1.70
	2025-04	€1.70
Primary IPv4 Total		€31.52
Setup EX44 Dedicated Server	2024-10	€390.00
	2024-12	€39.00
Setup EX44 Dedicated Server Total		€429.00
TOTAL		€1,246.49
Amazon Web Services S3 Storage for Datasets (USD)		
Storage	2024-09	$6.48
	2024-10	$21.96
	2024-11	$87.25
	2024-12	$97.83
	2025-01	$97.91
	2025-02	$97.96
	2025-03	$97.96
	2025-04	$97.97
	2025-05	$6.32
Storage Total		$611.64
Requests	2024-09	$9.91
	2024-10	$48.24
	2024-11	$104.08
	2024-12	$16.70
	2025-01	$9.67
	2025-04	$11.37
	2025-05	$0.04
Requests Total		$200.03
Transfer Out	2024-09	$0.00
	2025-04	$30.73
Transfer Out Total		$30.73
TOTAL		$842.40
GRAND TOTAL		**$8,513.57 + €1,246.49**
GRAND TOTAL IN GBP*		**£7,424.40**

Table A.1: Financial costs associated with the project broken down by service, charge type, and month.

* May 6 2025 rate

A.8 Open-source dependencies

With gratitude, I publish below a list of open-source software that I directly relied on, recognizing that their own dependencies deserve just as much praise. The ★ symbol indicates those dependencies that I considered crucial for my project. I wish to especially thank Felix Becker, the main contributor of `dom-to-svg`, the only crucial dependency that does not appear to be backed by a for-profit entity.

Python	
bert_score	https://github.com/Tiiiger/bert_score
bitsandbytes	https://github.com/bitsandbytes-foundation/bitsandbytes
boto3	https://github.com/boto/boto3
cssutils	https://github.com/jaraco/cssutils
flask	https://github.com/pallets/flask/
greenlet	https://greenlet.readthedocs.io/
helm	http://helm.last-exile.org/
★ HuggingFace accelerate	https://github.com/huggingface/accelerate
★ HuggingFace datasets	https://github.com/huggingface/datasets
★ HuggingFace evaluate	https://github.com/huggingface/evaluate
HuggingFace hub	https://github.com/huggingface/huggingface_hub
★ Huggingface peft	https://github.com/huggingface/peft
★ HuggingFace tokenizers	https://github.com/huggingface/tokenizers
★ HuggingFace transformers	https://github.com/huggingface/transformers
★ HuggingFace trl	https://github.com/huggingface/trl
★ lpips	https://github.com/richzhang/PerceptualSimilarity
lxml	https://github.com/lxml/lxml
matplotlib	https://matplotlib.org
numpy	https://numpy.org/
pandas	https://github.com/pandas-dev/pandas
pcfg	https://github.com/thomasbreydo/pcfg
Pillow	https://github.com/python-pillow/Pillow
★ Playwright	https://playwright.dev/
pyiqa	https://github.com/chaofengc/IQA-PyTorch
★ PyTorch	https://pytorch.org/
requests	https://github.com/psf/requests
scipy	https://scipy.org/
seaborn	https://github.com/mwaskom/seaborn
tbparse	https://github.com/j3soon/tbparse
tensorboard	https://github.com/tensorflow/tensorboard
torchvision	https://github.com/pytorch/vision
tqdm	https://github.com/tqdm/tqdm
★ unsloth	https://unsloth.ai/
vllm	https://github.com/vllm-project/vllm
NodeJS	
aws-sdk	https://aws.amazon.com/sdk-for-javascript/
★ Crawlee	https://crawlee.dev/
★ dom-to-svg (forked)	https://github.com/felixfbecker/dom-to-svg
express	https://expressjs.com/
HTMLMinifier	https://github.com/kangax/html-minifier
mime-types	https://github.com/jshttp/mime-types
★ Puppeteer	https://github.com/puppeteer/puppeteer
PurgeCSS	https://purgecss.com/
robots-txt-parser (forked)	https://github.com/ChrisAkroyd/robots-txt-parser
traverse	https://github.com/ljharb/js-traverse
xmldoc	https://github.com/nfarina/xmldoc

B
Word and Page Count

B.1 Word count

Chapter name	Words in text	Words headers	Words in captions	Total (without captions)
Introduction	754	3	28	757
Background	1,337	22	17	1,359
Goals	1,033	21	0	1,054
Data	4,991	53	445	5,044
Metrics	1,689	12	265	1,701
Models and training	6,930	112	790	7,042
Results	1,547	7	388	1,554
Conclusions	1063	7	14	1,070
TOTAL (without Appendix)	19,344	237	1,947	**19,581**
Appendix	3,208	31	57	3,239

I considered **19,581** the total wordcount from the perspective of the regulations: "As stated in the regulations, they must not exceed 20,000 words in length, excluding diagrams and appendices."

B.2 Page count

Total page count: 110.

References

[1] Josselin Somerville Roberts et al. *Image2Struct: Benchmarking Structure Extraction for Vision-Language Models*. arXiv:2410.22456 [cs]. Oct. 2024. DOI: 10.48550/arXiv.2410.22456. URL: http://arxiv.org/abs/2410.22456 (retrieved 5/4/2025).

[2] Yi Gui et al. 'WebCode2M: A Real-World Dataset for Code Generation from Webpage Designs'. In: *Proceedings of the ACM on Web Conference 2025*. arXiv:2404.06369 [cs]. Apr. 2025, pp. 1834–1845. DOI: 10.1145/3696410.3714889. URL: http://arxiv.org/abs/2404.06369 (retrieved 7/5/2025).

[3] Stack Overflow. *2024 Stack Overflow Developer Survey*. Publisher: Stack Overflow, Inc. 2024. URL: https://survey.stackoverflow.co/2024/ (retrieved 3/5/2025).

[4] Anonymous. *Most popular StackOverflow tags in 2024*. Publisher: Stack Exchange Data Explorer. 2024. URL: https://data.stackexchange.com/stackoverflow/query/1898980/most-popular-stackoverflow-tags-in-2024 (retrieved 3/5/2025).

[5] Stack Overflow. *2022 Stack Overflow Developer Survey — Most Loved, Dreaded, and Wanted Languages*. Publisher: Stack Overflow, Inc. 2022. URL: https://survey.stackoverflow.co/2022/#most-loved-dreaded-and-wanted-language-love-dread (retrieved 3/5/2025).

[6] Ashish Vaswani et al. *Attention Is All You Need*. arXiv:1706.03762 [cs]. June 2017. DOI: 10.48550/arXiv.1706.03762. URL: http://arxiv.org/abs/1706.03762 (retrieved 28/2/2025).

[7] Justin D. Weisz et al. *Examining the Use and Impact of an AI Code Assistant on Developer Productivity and Experience in the Enterprise*. arXiv:2412.06603 [cs]. Mar. 2025. DOI: 10.48550/arXiv.2412.06603. URL: http://arxiv.org/abs/2412.06603 (retrieved 5/5/2025).

[8] Kyle Daigle and GitHub Staff. *Survey: The AI Wave Continues to Grow on Software Development Teams*. Publisher: GitHub, Inc. Aug. 2024. URL: https://github.blog/news-insights/research/survey-ai-wave-grows/ (retrieved 5/5/2025).

[9] Tommy Geoco, Taylor Palmer and Jordan Singer. *2023 Design Tools Survey: UI Design*. 2023. URL: https://uxtools.co/survey/2023/ui-design/.

[10] Bert Bos. *Cascading Style Sheets Level 2 Revision 2 (CSS 2.2) Specification*. Apr. 2016. URL: https://www.w3.org/TR/CSS22/.

[11] Amelia Bellamy-Royds et al. *Scalable Vector Graphics (SVG) 2*. URL: https://www.w3.org/TR/SVG2/ (retrieved 3/3/2025).

[12] Hugo Touvron et al. *LLaMA: Open and Efficient Foundation Language Models*. arXiv:2302.13971 [cs]. Feb. 2023. DOI: 10.48550/arXiv.2302.13971. URL: http://arxiv.org/abs/2302.13971 (retrieved 5/5/2025).

[13] DeepSeek-AI et al. *DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning*. arXiv:2501.12948 [cs]. Jan. 2025. DOI: 10.48550/arXiv.2501.12948. URL: http://arxiv.org/abs/2501.12948 (retrieved 5/5/2025).

[14] Tony Beltramelli. *pix2code: Generating Code from a Graphical User Interface Screenshot*. arXiv:1705.07962 [cs]. Sept. 2017. DOI: 10.48550/arXiv.1705.07962. URL: http://arxiv.org/abs/1705.07962 (retrieved 28/2/2025).

[15] Zijian Ding et al. *Frontend Diffusion: Empowering Self-Representation of Junior Researchers and Designers Through Agentic Workflows*. arXiv:2502.03788 [cs]. Feb. 2025. DOI: 10.48550/arXiv.2502.03788. URL: http://arxiv.org/abs/2502.03788 (retrieved 5/5/2025).

[16] Alex Robinson. *Sketch2code: Generating a website from a paper mockup*. arXiv:1905.13750 [cs]. May 2019. DOI: 10.48550/arXiv.1905.13750. URL: http://arxiv.org/abs/1905.13750 (retrieved 28/2/2025).

[17] Kenton Lee et al. *Pix2Struct: Screenshot Parsing as Pretraining for Visual Language Understanding*. arXiv:2210.03347 [cs]. June 2023. DOI: 10.48550/arXiv.2210.03347. URL: http://arxiv.org/abs/2210.03347 (retrieved 28/2/2025).

[18] Geewook Kim et al. *OCR-free Document Understanding Transformer*. arXiv:2111.15664 [cs]. Oct. 2022. DOI: 10.48550/arXiv.2111.15664. URL: http://arxiv.org/abs/2111.15664 (retrieved 28/2/2025).

[19] Davit Soselia, Khalid Saifullah and Tianyi Zhou. *Learning UI-to-Code Reverse Generator Using Visual Critic Without Rendering*. arXiv:2305.14637 [cs]. Nov. 2023. DOI: 10.48550/arXiv.2305.14637. URL: http://arxiv.org/abs/2305.14637 (retrieved 28/2/2025).

[20] Alec Radford et al. *Language Models are Unsupervised Multitask Learners*. Feb. 2019. URL: https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.

[21] Yi Gui et al. 'UICopilot: Automating UI Synthesis via Hierarchical Code Generation from Webpage Designs'. en. In: *Proceedings of the ACM on Web Conference 2025*. Sydney NSW Australia: ACM, Apr. 2025, pp. 1846–1855. DOI: 10.1145/3696410.3714891. URL: https://dl.acm.org/doi/10.1145/3696410.3714891 (retrieved 5/5/2025).

[22] OpenAI et al. *GPT-4 Technical Report*. arXiv:2303.08774 [cs]. Mar. 2024. DOI: 10.48550/arXiv.2303.08774. URL: http://arxiv.org/abs/2303.08774 (retrieved 5/5/2025).

[23] Hong Jun Jeon and Benjamin Van Roy. *Information-Theoretic Foundations for Machine Learning*. arXiv:2407.12288 [stat]. Aug. 2024. DOI: 10.48550/arXiv.2407.12288. URL: http://arxiv.org/abs/2407.12288 (retrieved 3/3/2025).

[24] *Render-tree Construction, Layout, and Paint | Articles*. en. URL: https://web.dev/articles/critical-rendering-path/render-tree-construction (retrieved 3/3/2025).

[25] Martijn Koster et al. *Robots Exclusion Protocol*. Issue: 9309 Num Pages: 12 Series: Request for Comments Published: RFC 9309. Sept. 2022. DOI: 10.17487/RFC9309. URL: https://www.rfc-editor.org/info/rfc9309.

[26] Yi Tay et al. *Long Range Arena: A Benchmark for Efficient Transformers*. en. arXiv:2011.04006 [cs]. Nov. 2020. DOI: 10.48550/arXiv.2011.04006. URL: http://arxiv.org/abs/2011.04006 (retrieved 4/3/2025).

[27] Jordan Hoffmann et al. *Training Compute-Optimal Large Language Models*. en. arXiv:2203.15556 [cs]. Mar. 2022. DOI: 10.48550/arXiv.2203.15556. URL: http://arxiv.org/abs/2203.15556 (retrieved 5/3/2025).

[28] Niklas Muennighoff et al. *Scaling Data-Constrained Language Models*. en. arXiv:2305.16264 [cs]. Oct. 2023. DOI: 10.48550/arXiv.2305.16264. URL: http://arxiv.org/abs/2305.16264 (retrieved 6/3/2025).

[29] Biao Zhang et al. *When Scaling Meets LLM Finetuning: The Effect of Data, Model and Finetuning Method*. en. arXiv:2402.17193 [cs]. Feb. 2024. DOI: 10.48550/arXiv.2402.17193. URL: http://arxiv.org/abs/2402.17193 (retrieved 5/3/2025).

[30] Proyag Pal, Alexandra Birch and Kenneth Heafield. 'Document-Level Machine Translation with Large-Scale Public Parallel Corpora'. In: *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Lun-Wei Ku, Andre Martins and Vivek Srikumar. Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024, pp. 13185–13197. URL: https://aclanthology.org/2024.acl-long.712.

[31] Biao Zhang et al. 'Improving Massively Multilingual Neural Machine Translation and Zero-Shot Translation'. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Ed. by Dan Jurafsky et al. Online: Association for Computational Linguistics, July 2020, pp. 1628–1639. DOI: 10.18653/v1/2020.acl-main.148. URL: https://aclanthology.org/2020.acl-main.148.

[32] Jörg Tiedemann. 'Parallel Data, Tools and Interfaces in OPUS'. In: *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*. Ed. by Nicoletta Calzolari et al. Istanbul, Turkey: European Language Resources Association (ELRA), May 2012, pp. 2214–2218. URL: http://www.lrec-conf.org/proceedings/lrec2012/pdf/463_Paper.pdf.

[33] Vukosi Marivate et al. *The Vuk'uzenzele South African Multilingual Corpus*. Feb. 2023. DOI: 10.5281/zenodo.7598539. URL: https://doi.org/10.5281/zenodo.7598539.

[34] Isaac Caswell et al. *SMOL: Professionally translated parallel data for 115 under-represented languages*. _eprint: 2502.12301. 2025. URL: https://arxiv.org/abs/2502.12301.

[35] *Jinja*. URL: https://github.com/pallets/jinja.

[36] Alessio Miaschi et al. 'What Makes My Model Perplexed? A Linguistic Investigation on Neural Language Models Perplexity'. en. In: *Proceedings of Deep Learning Inside Out (DeeLIO): The 2nd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures*. Online: Association for Computational Linguistics, 2021, pp. 40–47. DOI: 10.18653/v1/2021.deelio-1.5. URL: https://www.aclweb.org/anthology/2021.deelio-1.5 (retrieved 11/3/2025).

[37] T.L. Booth and R.A. Thompson. 'Applying Probability Measures to Abstract Languages'. In: *IEEE Transactions on Computers* C-22.5 (May 1973), pp. 442–450. DOI: 10.1109/T-C.1973.223746. URL: http://ieeexplore.ieee.org/document/1672339/ (retrieved 11/3/2025).

[38] Dolav Nitay, Dana Fisman and Michal Ziv-Ukelson. *Learning of Structurally Unambiguous Probabilistic Grammars*. en. arXiv:2011.07472 [cs]. Mar. 2021. DOI: 10.48550/arXiv.2011.07472. URL: http://arxiv.org/abs/2011.07472 (retrieved 11/3/2025).

[39] Mistral AI. *Codestral*. URL: https://mistral.ai/news/codestral.

[40] Shengyu Zhang et al. *Instruction Tuning for Large Language Models: A Survey*. en. arXiv:2308.10792 [cs]. Dec. 2024. DOI: 10.48550/arXiv.2308.10792. URL: http://arxiv.org/abs/2308.10792 (retrieved 12/3/2025).

[41] Hongbin Ye et al. *Cognitive Mirage: A Review of Hallucinations in Large Language Models*. en. arXiv:2309.06794 [cs]. Sept. 2023. DOI: 10.48550/arXiv.2309.06794. URL: http://arxiv.org/abs/2309.06794 (retrieved 12/3/2025).

[42] Tianyi Zhang et al. *BERTScore: Evaluating Text Generation with BERT*. en. arXiv:1904.09675 [cs]. Feb. 2020. DOI: 10.48550/arXiv.1904.09675. URL: http://arxiv.org/abs/1904.09675 (retrieved 12/3/2025).

[43] Common Crawl. *Common Crawl Dataset*. 2024. URL: https://commoncrawl.org.

[44] Henrik Nielsen et al. *Hypertext Transfer Protocol – HTTP/1.1*. Issue: 2616 Num Pages: 176 Series: Request for Comments Published: RFC 2616. June 1999. DOI: 10.17487/RFC2616. URL: https://www.rfc-editor.org/info/rfc2616.

[45] Charlie Snell et al. *Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters*. arXiv:2408.03314 [cs]. Aug. 2024. DOI: 10.48550/arXiv.2408.03314. URL: http://arxiv.org/abs/2408.03314 (retrieved 3/5/2025).

[46] Chaofeng Chen et al. *TOPIQ: A Top-down Approach from Semantics to Distortions for Image Quality Assessment*. arXiv:2308.03060 [cs]. Aug. 2023. DOI: 10.48550/arXiv.2308.03060. URL: http://arxiv.org/abs/2308.03060 (retrieved 2/4/2025).

[47] Shanshan Lao et al. *Attentions Help CNNs See Better: Attention-based Hybrid Image Quality Assessment Network*. arXiv:2204.10485 [cs]. Apr. 2022. DOI: 10.48550/arXiv.2204.10485. URL: http://arxiv.org/abs/2204.10485 (retrieved 2/4/2025).

[48] Ekta Prashnani et al. *PieAPP: Perceptual Image-Error Assessment through Pairwise Preference*. arXiv:1806.02067 [cs]. June 2018. DOI: 10.48550/arXiv.1806.02067. URL: http://arxiv.org/abs/1806.02067 (retrieved 2/4/2025).

[49] Richard Zhang et al. *The Unreasonable Effectiveness of Deep Features as a Perceptual Metric*. arXiv:1801.03924 [cs]. Apr. 2018. DOI: 10.48550/arXiv.1801.03924. URL: http://arxiv.org/abs/1801.03924 (retrieved 2/4/2025).

[50] Keyan Ding et al. 'Image Quality Assessment: Unifying Structure and Texture Similarity'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2020). arXiv:2004.07728 [cs], pp. 1–1. DOI: 10.1109/TPAMI.2020.3045810. URL: http://arxiv.org/abs/2004.07728 (retrieved 2/4/2025).

[51] Sebastian Bosse et al. 'Deep Neural Networks for No-Reference and Full-Reference Image Quality Assessment'. In: *IEEE Transactions on Image Processing* 27.1 (Jan. 2018). Conference Name: IEEE Transactions on Image Processing, pp. 206–219. DOI: 10.1109/TIP.2017.2760518. URL: https://ieeexplore.ieee.org/document/8063957 (retrieved 2/4/2025).

[52] Heliang Zheng et al. *Learning Conditional Knowledge Distillation for Degraded-Reference Image Quality Assessment*. arXiv:2108.07948 [eess]. Aug. 2021. DOI: 10.48550/arXiv.2108.07948. URL: http://arxiv.org/abs/2108.07948 (retrieved 2/4/2025).

[53] Lin Zhang et al. 'FSIM: A Feature Similarity Index for Image Quality Assessment'. In: *IEEE Transactions on Image Processing* 20.8 (Aug. 2011), pp. 2378–2386. DOI: 10.1109/TIP.2011.2109730. URL: http://ieeexplore.ieee.org/document/5705575/ (retrieved 2/4/2025).

[54] Zhou Wang, Eero P. Simoncelli and Alan C. Bovik. 'Multi-Scale Structural Similarity for Image Quality Assessment'. In: *Proceedings of the 37th IEEE Asilomar Conference on Signals, Systems and Computers*. Pacific Grove, CA: IEEE, Nov. 2003, pp. 1398–1402. URL: https://ieeexplore.ieee.org/document/1292216.

[55] M.P. Sampat et al. 'Complex Wavelet Structural Similarity: A New Image Similarity Index'. In: *IEEE Transactions on Image Processing* 18.11 (Nov. 2009), pp. 2385–2401. DOI: 10.1109/TIP.2009.2025923. URL: http://ieeexplore.ieee.org/document/5109651/ (retrieved 2/4/2025).

[56] H.R. Sheikh and A.C. Bovik. 'Image information and visual quality'. In: *IEEE Transactions on Image Processing* 15.2 (Feb. 2006), pp. 430–444. DOI: 10.1109/TIP.2005.859378. URL: https://ieeexplore.ieee.org/document/1576816 (retrieved 2/4/2025).

[57] Wufeng Xue et al. 'Gradient Magnitude Similarity Deviation: A Highly Efficient Perceptual Image Quality Index'. In: *IEEE Transactions on Image Processing* 23.2 (Feb. 2014). arXiv:1308.3052 [cs], pp. 684–695. DOI: 10.1109/TIP.2013.2293423. URL: http://arxiv.org/abs/1308.3052 (retrieved 2/4/2025).

[58] Valero Laparra et al. 'Perceptual image quality assessment using a normalized Laplacian pyramid'. In: *Electronic Imaging* 28.16 (Feb. 2016), pp. 1–6. DOI: 10.2352/ISSN.2470-1173.2016.16.HVEI-103. URL: https://library.imaging.org/ei/articles/28/16/art00008 (retrieved 2/4/2025).

[59] Lin Zhang, Ying Shen and Hongyu Li. 'VSI: A Visual Saliency-Induced Index for Perceptual Image Quality Assessment'. In: *IEEE Transactions on Image Processing* 23.10 (Oct. 2014), pp. 4270–4281. DOI: 10.1109/TIP.2014.2346028. URL: https://ieeexplore.ieee.org/document/6873260/ (retrieved 2/4/2025).

[60] Allan G. Weber. *The USC-SIPI Image Database: Version 6*. Tech. rep. 432. Los Angeles, CA: Signal and Image Processing Institute, University of Southern California, Feb. 2018. URL: https://sipi.usc.edu/database/SIPI_Database.pdf.

[61] Zhou Wang et al. 'Image quality assessment: from error visibility to structural similarity'. In: *IEEE Transactions on Image Processing* 13.4 (Apr. 2004), pp. 600–612. DOI: 10.1109/TIP.2003.819861. URL: https://ieeexplore.ieee.org/document/1284395/ (retrieved 5/4/2025).

[62] Kishore Papineni et al. 'BLEU: a method for automatic evaluation of machine translation'. en. In: *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics - ACL '02*. Philadelphia, Pennsylvania: Association for Computational Linguistics, 2001, p. 311. DOI: 10.3115/1073083.1073135. URL: http://portal.acm.org/citation.cfm?doid=1073083.1073135 (retrieved 5/4/2025).

[63] Alec Radford et al. *Learning Transferable Visual Models From Natural Language Supervision*. arXiv:2103.00020 [cs]. Feb. 2021. DOI: 10.48550/arXiv.2103.00020. URL: http://arxiv.org/abs/2103.00020 (retrieved 8/5/2025).

[64] Y. Rubner, C. Tomasi and L.J. Guibas. 'A metric for distributions with applications to image databases'. In: *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*. Bombay, India: Narosa Publishing House, 1998, pp. 59–66. DOI: 10.1109/ICCV.1998.710701. URL: http://ieeexplore.ieee.org/document/710701/ (retrieved 5/4/2025).

[65] Chun-Fu Chen, Quanfu Fan and Rameswar Panda. *CrossViT: Cross-Attention Multi-Scale Vision Transformer for Image Classification*. en. arXiv:2103.14899 [cs]. Aug. 2021. DOI: 10.48550/arXiv.2103.14899. URL: http://arxiv.org/abs/2103.14899 (retrieved 11/4/2025).

[66] Ceder Dens et al. *A cross-attention transformer encoder for paired sequence data*. en. Dec. 2023. DOI: 10.1101/2023.12.11.571066. URL: http://biorxiv.org/lookup/doi/10.1101/2023.12.11.571066 (retrieved 11/4/2025).

[67] A. Hammad, S. Moretti and M. Nojiri. *Multi-scale cross-attention transformer encoder for event classification*. arXiv:2401.00452 [hep-ph]. Feb. 2024. DOI: 10.48550/arXiv.2401.00452. URL: http://arxiv.org/abs/2401.00452 (retrieved 11/4/2025).

[68] Tom Kocmi et al. *Preliminary WMT24 Ranking of General MT Systems and LLMs*. arXiv:2407.19884 [cs]. July 2024. DOI: 10.48550/arXiv.2407.19884. URL: http://arxiv.org/abs/2407.19884 (retrieved 11/4/2025).

[69] Chen Yang et al. *The Fine Line: Navigating Large Language Model Pretraining with Down-streaming Capability Analysis*. arXiv:2404.01204 [cs]. Nov. 2024. DOI: 10.48550/arXiv.2404.01204. URL: http://arxiv.org/abs/2404.01204 (retrieved 11/4/2025).

[70] Michael Han Daniel Han and Unsloth team. *Unsloth*. 2023. URL: http://github.com/unslothai/unsloth.

[71] Yi Tay et al. *Efficient Transformers: A Survey*. arXiv:2009.06732 [cs]. Mar. 2022. DOI: 10.48550/arXiv.2009.06732. URL: http://arxiv.org/abs/2009.06732 (retrieved 13/4/2025).

[72] Tri Dao et al. *FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness*. arXiv:2205.14135 [cs]. June 2022. DOI: 10.48550/arXiv.2205.14135. URL: http://arxiv.org/abs/2205.14135 (retrieved 13/4/2025).

[73] Woosuk Kwon et al. *Efficient Memory Management for Large Language Model Serving with PagedAttention*. arXiv:2309.06180 [cs]. Sept. 2023. DOI: 10.48550/arXiv.2309.06180. URL: http://arxiv.org/abs/2309.06180 (retrieved 13/4/2025).

[74] Albert Gu and Tri Dao. *Mamba: Linear-Time Sequence Modeling with Selective State Spaces*. arXiv:2312.00752 [cs]. May 2024. DOI: 10.48550/arXiv.2312.00752. URL: http://arxiv.org/abs/2312.00752 (retrieved 13/4/2025).

[75] Michael Poli et al. *Hyena Hierarchy: Towards Larger Convolutional Language Models*. arXiv:2302.10866 [cs]. Apr. 2023. DOI: 10.48550/arXiv.2302.10866. URL: http://arxiv.org/abs/2302.10866 (retrieved 13/4/2025).

[76] Colin Raffel et al. *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. arXiv:1910.10683 [cs]. Sept. 2023. DOI: 10.48550/arXiv.1910.10683. URL: http://arxiv.org/abs/1910.10683 (retrieved 14/4/2025).

[77] Thomas Wolf et al. *HuggingFace's Transformers: State-of-the-art Natural Language Processing*. arXiv:1910.03771 [cs]. July 2020. DOI: 10.48550/arXiv.1910.03771. URL: http://arxiv.org/abs/1910.03771 (retrieved 14/4/2025).

[78] Victor Sanh et al. *DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter*. arXiv:1910.01108 [cs]. Mar. 2020. DOI: 10.48550/arXiv.1910.01108. URL: http://arxiv.org/abs/1910.01108 (retrieved 16/4/2025).

[79] Zhilin Yang et al. *XLNet: Generalized Autoregressive Pretraining for Language Understanding*. arXiv:1906.08237 [cs]. Jan. 2020. DOI: 10.48550/arXiv.1906.08237. URL: http://arxiv.org/abs/1906.08237 (retrieved 16/4/2025).

[80] Iz Beltagy, Matthew E. Peters and Arman Cohan. *Longformer: The Long-Document Transformer*. arXiv:2004.05150 [cs]. Dec. 2020. DOI: 10.48550/arXiv.2004.05150. URL: http://arxiv.org/abs/2004.05150 (retrieved 16/4/2025).

[81] Nikita Kitaev, Łukasz Kaiser and Anselm Levskaya. *Reformer: The Efficient Transformer*. arXiv:2001.04451 [cs]. Feb. 2020. DOI: 10.48550/arXiv.2001.04451. URL: http://arxiv.org/abs/2001.04451 (retrieved 16/4/2025).

[82] Aurko Roy et al. *Efficient Content-Based Sparse Attention with Routing Transformers*. arXiv:2003.05997 [cs]. Oct. 2020. DOI: 10.48550/arXiv.2003.05997. URL: http://arxiv.org/abs/2003.05997 (retrieved 16/4/2025).

[83] Victor Sanh et al. *Multitask Prompted Training Enables Zero-Shot Task Generalization*. arXiv:2110.08207 [cs]. Mar. 2022. DOI: 10.48550/arXiv.2110.08207. URL: http://arxiv.org/abs/2110.08207 (retrieved 16/4/2025).

[84] Mandy Guo et al. *LongT5: Efficient Text-To-Text Transformer for Long Sequences*. arXiv:2112.07916 [cs]. May 2022. DOI: 10.48550/arXiv.2112.07916. URL: http://arxiv.org/abs/2112.07916 (retrieved 16/4/2025).

[85] Yue Wang et al. *CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation*. arXiv:2109.00859 [cs]. Sept. 2021. DOI: 10.48550/arXiv.2109.00859. URL: http://arxiv.org/abs/2109.00859 (retrieved 16/4/2025).

[86] Tao Ge, Si-Qing Chen and Furu Wei. *EdgeFormer: A Parameter-Efficient Transformer for On-Device Seq2seq Generation*. arXiv:2202.07959 [cs]. Dec. 2022. DOI: 10.48550/arXiv.2202.07959. URL: http://arxiv.org/abs/2202.07959 (retrieved 16/4/2025).

[87] Yue Wang et al. *CodeT5+: Open Code Large Language Models for Code Understanding and Generation*. arXiv:2305.07922 [cs]. May 2023. DOI: 10.48550/arXiv.2305.07922. URL: http://arxiv.org/abs/2305.07922 (retrieved 16/4/2025).

[88] Manzil Zaheer et al. *Big Bird: Transformers for Longer Sequences*. arXiv:2007.14062 [cs]. Jan. 2021. DOI: 10.48550/arXiv.2007.14062. URL: http://arxiv.org/abs/2007.14062 (retrieved 16/4/2025).

[89] Baptiste Rozière et al. *Code Llama: Open Foundation Models for Code*. arXiv:2308.12950 [cs]. Jan. 2024. DOI: 10.48550/arXiv.2308.12950. URL: http://arxiv.org/abs/2308.12950 (retrieved 16/4/2025).

[90] Aaron Grattafiori et al. *The Llama 3 Herd of Models*. arXiv:2407.21783 [cs]. Nov. 2024. DOI: 10.48550/arXiv.2407.21783. URL: http://arxiv.org/abs/2407.21783 (retrieved 16/4/2025).

[91] Qwen et al. *Qwen2.5 Technical Report*. arXiv:2412.15115 [cs]. Jan. 2025. DOI: 10.48550/arXiv.2412.15115. URL: http://arxiv.org/abs/2412.15115 (retrieved 16/4/2025).

[92] Vast.ai Inc. *Vast.ai*. 2025. URL: https://vast.ai/ (retrieved 17/4/2025).

[93] Adam Paszke et al. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. arXiv:1912.01703 [cs]. Dec. 2019. DOI: 10.48550/arXiv.1912.01703. URL: http://arxiv.org/abs/1912.01703 (retrieved 17/4/2025).

[94] Jason Ansel et al. 'PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation'. In: *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*. ACM, Apr. 2024. DOI: 10.1145/3620665.3640366. URL: https://pytorch.org/assets/pytorch2-2.pdf.

[95] Quentin Lhoest et al. *Datasets: A Community Library for Natural Language Processing*. arXiv:2109.02846 [cs]. Sept. 2021. DOI: 10.48550/arXiv.2109.02846. URL: http://arxiv.org/abs/2109.02846 (retrieved 17/4/2025).

[96] Leandro von Werra et al. *Evaluate & Evaluation on the Hub: Better Best Practices for Data and Model Measurements*. arXiv:2210.01970 [cs]. Oct. 2022. DOI: 10.48550/arXiv.2210.01970. URL: http://arxiv.org/abs/2210.01970 (retrieved 17/4/2025).

[97] Kluyver Thomas et al. 'Jupyter Notebooks – a publishing format for reproducible computational workflows'. In: *Positioning and Power in Academic Publishing: Players, Agents and Agendas*. IOS Press, 2016. DOI: 10.3233/978-1-61499-649-1-87. URL: https://www.medra.org/servlet/aliasResolver?alias=iospressISBN&isbn=978-1-61499-648-4&spage=87&doi=10.3233/978-1-61499-649-1-87 (retrieved 18/4/2025).

[98] D. E. Knuth. 'Literate Programming'. en. In: *The Computer Journal* 27.2 (Feb. 1984), pp. 97–111. DOI: 10.1093/comjnl/27.2.97. URL: https://academic.oup.com/comjnl/article-lookup/doi/10.1093/comjnl/27.2.97 (retrieved 18/4/2025).

[99] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems*. arXiv:1603.04467 [cs]. Mar. 2016. DOI: 10.48550/arXiv.1603.04467. URL: http://arxiv.org/abs/1603.04467 (retrieved 18/4/2025).

[100] Taku Kudo and John Richardson. *SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing*. arXiv:1808.06226 [cs]. Aug. 2018. DOI: 10.48550/arXiv.1808.06226. URL: http://arxiv.org/abs/1808.06226 (retrieved 24/4/2025).

[101] Sean McLeish et al. *Transformers Can Do Arithmetic with the Right Embeddings*. arXiv:2405.17399 [cs]. Dec. 2024. DOI: 10.48550/arXiv.2405.17399. URL: http://arxiv.org/abs/2405.17399 (retrieved 5/5/2025).

[102] Nayoung Lee et al. *Teaching Arithmetic to Small Transformers*. arXiv:2307.03381 [cs]. July 2023. DOI: 10.48550/arXiv.2307.03381. URL: http://arxiv.org/abs/2307.03381 (retrieved 5/5/2025).

[103] Tianyi Zhou et al. *Pre-trained Large Language Models Use Fourier Features to Compute Addition*. arXiv:2406.03445 [cs]. June 2024. DOI: 10.48550/arXiv.2406.03445. URL: http://arxiv.org/abs/2406.03445 (retrieved 5/5/2025).

[104] Hyung Won Chung et al. *Scaling Instruction-Finetuned Language Models*. arXiv:2210.11416 [cs]. Dec. 2022. DOI: 10.48550/arXiv.2210.11416. URL: http://arxiv.org/abs/2210.11416 (retrieved 25/4/2025).

[105] Ilya Loshchilov and Frank Hutter. *Decoupled Weight Decay Regularization*. arXiv:1711.05101 [cs]. Jan. 2019. DOI: 10.48550/arXiv.1711.05101. URL: http://arxiv.org/abs/1711.05101 (retrieved 30/4/2025).

[106] Tri Dao. *FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning*. arXiv:2307.08691 [cs]. July 2023. DOI: 10.48550/arXiv.2307.08691. URL: http://arxiv.org/abs/2307.08691 (retrieved 30/4/2025).

[107] Edward J. Hu et al. *LoRA: Low-Rank Adaptation of Large Language Models*. arXiv:2106.09685 [cs]. Oct. 2021. DOI: 10.48550/arXiv.2106.09685. URL: http://arxiv.org/abs/2106.09685 (retrieved 30/4/2025).

[108] Tim Dettmers et al. *QLoRA: Efficient Finetuning of Quantized LLMs*. arXiv:2305.14314 [cs]. May 2023. DOI: 10.48550/arXiv.2305.14314. URL: http://arxiv.org/abs/2305.14314 (retrieved 1/5/2025).

[109] Damjan Kalajdzievski. *A Rank Stabilization Scaling Factor for Fine-Tuning with LoRA*. arXiv:2312.03732 [cs]. Nov. 2023. DOI: 10.48550/arXiv.2312.03732. URL: http://arxiv.org/abs/2312.03732 (retrieved 1/5/2025).

[110] Hongrong Cheng, Miao Zhang and Javen Qinfeng Shi. *A Survey on Deep Neural Network Pruning-Taxonomy, Comparison, Analysis, and Recommendations*. arXiv:2308.06767 [cs]. Aug. 2024. DOI: 10.48550/arXiv.2308.06767. URL: http://arxiv.org/abs/2308.06767 (retrieved 3/5/2025).

[111] Yi Su et al. *Crossing the Reward Bridge: Expanding RL with Verifiable Rewards Across Diverse Domains*. arXiv:2503.23829 [cs]. Apr. 2025. DOI: 10.48550/arXiv.2503.23829. URL: http://arxiv.org/abs/2503.23829 (retrieved 8/5/2025).

[112] David J. Anderson. *Kanban: Successful Evolutionary Change for Your Technology Business*. Sequim, WA: Blue Hole Press, 2010.

[113] Francesco Cirillo. *The Pomodoro Technique: The Acclaimed Time-Management System That Has Transformed How We Work*. New York: Crown Currency, 2018.

[114] Martin Heidegger. 'The Question Concerning Technology'. In: *Basic Writings*. Ed. by David Farrell Krell. Trans. by William Lovitt. New York: Harper & Row, 1977, pp. 287–317.

[115] World Economic Forum. *The Future of Jobs Report 2020*. Insight Report. World Economic Forum, Oct. 2020. URL: https://www3.weforum.org/docs/WEF_Future_of_Jobs_2020.pdf (retrieved 3/5/2025).

[116] World Economic Forum. *Jobs of Tomorrow: Large Language Models and Jobs*. White Paper. World Economic Forum, Sept. 2023. URL: https://www3.weforum.org/docs/WEF_Jobs_of_Tomorrow_Generative_AI_2023.pdf (retrieved 3/5/2025).

[117] Jaron Lanier. 'There Is No A.I.' In: *The New Yorker* (Apr. 2023). URL: https://www.newyorker.com/science/annals-of-artificial-intelligence/there-is-no-ai (retrieved 28/4/2025).

[118] Bret Victor. *Bret Victor — Worrydream*. 2025. URL: https://worrydream.com/ (retrieved 3/5/2025).

[119] Robert M. Pirsig. *Zen and the Art of Motorcycle Maintenance: An Inquiry into Values*. New York: William Morrow & Company, 1974. URL: https://en.wikipedia.org/wiki/Zen_and_the_Art_of_Motorcycle_Maintenance.

[120] Shayne Longpre et al. *Consent in Crisis: The Rapid Decline of the AI Data Commons*. arXiv:2407.14933 [cs]. July 2024. DOI: 10.48550/arXiv.2407.14933. URL: http://arxiv.org/abs/2407.14933 (retrieved 5/5/2025).

[121] Alexander Song, Dayuan Chen and Ziliang Zong. 'Unveiling the Truth: An Analysis of the Energy and Carbon Footprint of Training an OPT Model using DeepSpeed on the H100 GPU'. en. In: *Proceedings of the 14th International Green and Sustainable Computing Conference*. Toronto ON Canada: ACM, Oct. 2023, pp. 36–38. DOI: 10.1145/3634769.3634806. URL: https://dl.acm.org/doi/10.1145/3634769.3634806 (retrieved 3/5/2025).

[122] *The New York Times Company v. Microsoft Corp. & OpenAI*. Published: U.\,S. District Court, Southern District of New York, No.\ 1:23-cv-11195-SHS. 2023. URL: https://www.courtlistener.com/docket/68117049/the-new-york-times-company-v-microsoft-corporation/.

[123] *OpenAI, Inc. Copyright Infringement Litigation*. Published: U.,S. District Court, Southern District of New York, MDL No. 1:25-md-03143-SHS. 2025. URL: https://www.courtlistener.com/docket/69879510/in-re-openai-inc-copyright-infringement-litigation/.

[124] *Kadrey v. Meta Platforms, Inc.* Published: U.,S. District Court, Northern District of California, No. 3:23-cv-03417-VC. 2023. URL: https://www.courtlistener.com/docket/67569326/kadrey-v-meta-platforms-inc/.

[125] *Doe v. GitHub, Inc. et al.* Published: U.,S. District Court, Northern District of California, No. 3:22-cv-06823. 2022. URL: https://www.law360.com/cases/636417914eba833b62f13a4e.

[126] *Getty Images (US), Inc. v. Stability AI, Inc.* Published: U.,S. District Court, District of Delaware, No. 1:23-cv-00135. 2023. URL: https://www.courtlistener.com/docket/66788385/getty-images-us-inc-v-stability-ai-inc/.

[127] *Andersen v. Stability AI Ltd. et al.* Published: U.,S. District Court, Northern District of California, No. 3:23-cv-00201-WHO. 2023. URL: https://www.courtlistener.com/docket/66732129/andersen-v-stability-ai-ltd/.

[128] U.S. Congress. *17 U.S. Code § 107 - Limitations on exclusive rights: Fair use*. 2024. URL: https://www.law.cornell.edu/uscode/text/17/107.

[129] *Authors Guild, Inc. v. Google, Inc.* 2015.

[130] European Parliament and Council. *Directive (EU) 2019/790 of the European Parliament and of the Council of 17 April 2019 on copyright and related rights in the Digital Single Market*. 2019. URL: https://eur-lex.europa.eu/eli/dir/2019/790/oj/eng.

[131] European Parliament and Council. *Regulation (EU) 2024/1689 of the European Parliament and of the Council of 13 June 2024 on the transparency and fairness of copyright contracts*. 2024. URL: https://eur-lex.europa.eu/eli/reg/2024/1689/oj/eng.

[132] UK Parliament. *Copyright, Designs and Patents Act 1988*. 1988. URL: https://www.legislation.gov.uk/ukpga/1988/48/contents.

[133] UK Government. *Copyright and Artificial Intelligence: Call for Views*. 2021. URL: https://www.gov.uk/government/consultations/copyright-and-artificial-intelligence/copyright-and-artificial-intelligence.