Generative Colorization of Structured Mobile Web Pages

Kotaro Kikuchi[1] Naoto Inoue[1] Mayu Otani[1] Edgar Simo-Serra[2] Kota Yamaguchi[1]
[1]CyberAgent [2]Waseda University

Abstract

Color is a critical design factor for web pages, affecting important factors such as viewer emotions and the overall trust and satisfaction of a website. Effective coloring requires design knowledge and expertise, but if this process could be automated through data-driven modeling, efficient exploration and alternative workflows would be possible. However, this direction remains underexplored due to the lack of a formalization of the web page colorization problem, datasets, and evaluation protocols. In this work, we propose a new dataset consisting of e-commerce mobile web pages in a tractable format, which are created by simplifying the pages and extracting canonical color styles with a common web browser. The web page colorization problem is then formalized as a task of estimating plausible color styles for a given web page content with a given hierarchical structure of the elements. We present several Transformer-based methods that are adapted to this task by prepending structural message passing to capture hierarchical relationships between elements. Experimental results, including a quantitative evaluation designed for this task, demonstrate the advantages of our methods over statistical and image colorization methods. The code is available at `https://github.com/CyberAgentAILab/webcolor`.

1. Introduction

Color plays an important role in the visual communication of web pages. It is known that colors are associated with certain emotions [15] and have a significant impact on the trust and satisfaction of websites [7]. Effectively coloring web pages to achieve the design goals is a difficult task, as it requires understanding the theories and heuristics about colors and their combinations [27]. Another reason for the difficulty lies in the implicit and practical requirements within a page, such as overall balance, contrast, and differentiation of color connotations. Our aim is to overcome these difficulties with data-driven modeling to facilitate new applications such as efficient design exploration for designers, automated design for non-designers, and automated creation of landing pages for advertised products.

Despite its great industrial potential, the lack of established benchmarks and the need for extensive domain knowledge for data collection make it difficult to apply data-driven methodologies to web page coloring, which may explain why there are fewer related studies in the literature. Existing methods for coloring web pages either generate color styles for some specific elements [30, 25] or require already designed web pages [29, 11], and no method has been investigated that can add plausible color styles to all elements from scratch. Also, the datasets used in these studies are not publicly available, making follow-up studies difficult. The image colorization techniques [1, 19] can be applied to web page, however, they require an input image and the extraction of structural color information from the output image, which narrows the applicable scenarios and would cause extraction errors that degrade quality.

In this work, we construct a new dataset to take the study of web page coloring one step further. The challenges in data collection are that raw web pages have many redundant elements that do not affect their appearance, and the color styles defined in their styling sheets are ambiguous as the same color can be represented differently. To address these challenges, we use a common web browser to simplify web pages and retrieve the browser's resolved styles as canonical color styles.

The coloring problem is then formalized as a task of generating plausible color styles for web pages provided with content information and hierarchical structure of the elements. We present several Transformer-based methods adapted to this task. The hierarchical relationship between elements affects the color style, since child elements are basically rendered on top of their parent elements. Our hierarchical modeling to capture these characteristics is achieved by contextualizing the features with message passing [8] before feeding them into the Transformer network.

To evaluate the generated color styles in terms of quality and diversity, we adapt the metric used in image generation tasks [12] to ours and test whether the resulting pages satisfy the standard contrast criterion. Quantitative and qualitative results demonstrate the advantages of our methods over statistical and image colorization methods [19].

Our contributions are summarized as follows:

- A dataset of e-commerce mobile web pages tailored for automatic coloring of web pages.
- A problem formulation of the web page coloring and Transformer [28]-based methods that capture hierarchical relationships between elements through structural message passing.
- Extensive evaluation including comparison with a state-of-the-art image colorization method [19].

2. Related Work

2.1. Image Recoloring and Colorization

The problem of automatically determining color has been extensively addressed, especially for images. There are two common tasks: *recoloring*, which applies different colors to a color image, and *colorization*, which applies colors to a grayscale image. For both tasks, the reference-based approach has been investigated, which is achieved by finding a mapping between the color or luminance of the source and the color of a given reference image or palette [5, 24, 4, 17]. Volkova *et al*. [29] adapted this approach for recoloring a web page with a reference image.

The reference-based approach requires additional user input, whereas the approach that uses color knowledge learned from data does not. Earlier work in this direction used relatively simple models to learn cost functions, such as color compatibility, and then colorize images or 2D patterns through optimization [22, 21]. Recently, deep learning-based methods have been actively studied, especially for the image colorization task [1], and ColTran [19] that employs the Transformer network [28] as its base model has demonstrated state-of-the-art performance.

There are two concerns in applying image colorization techniques to web pages. The first one is that the input is an image, which narrows the applicable scenarios, and the second is that the quality could be degraded by errors in extracting the colors from the output image to be used for the web page. In our problem formulation, these concerns do not arise because we treat web page coloring as an estimation problem on structural data.

2.2. Data-driven Applications for Web Pages

Research on data-driven applications for web pages has been actively studied in recent years in different communities, including question answering [6], semantic labeling of elements [14], information retrieval [23], code generation from a screenshot [3], layout generation [16], and GUI photomontage [31]. An earlier work on data-driven web page styling is Webzeitgeist [20], a platform that provides several data and demographics for over 100,000 web pages, but is no longer publicly available.

Relatively few studies have been done for the automatic coloring. Gu *et al*. [11] propose an optimization-based coloring framework by combining several models, including a color contrast model learned using 500 web pages, and confirm its effectiveness in two tasks: coloring with a prepared palette and fine-tuning a color design. Zhao *et al*. [30] address the task of estimating font attributes of an element, including font color, from the surrounding context. Most recently, Qiu *et al*. [25] propose a Transformer-based color recommendation model for landing pages that is trained with the masked prediction and the advertisement performance prediction. Their model uses multiple color palettes extracted from specific regions of annotated screenshots and can recommend other colors for the target region.

In summary, there is no established method for adding plausible color styles to all elements from scratch. There is also no dataset available to study such a method, therefore, we begin with the construction of a dataset on the structural coloring of web pages.

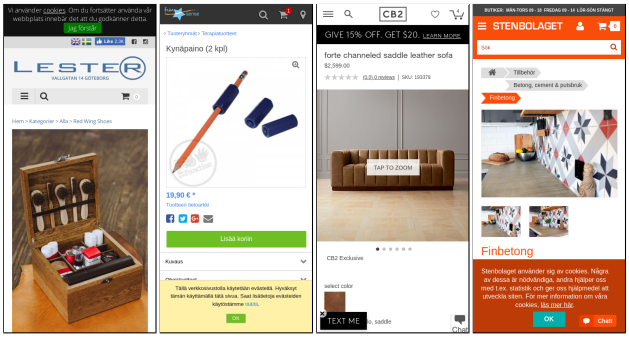

Figure 1: Screenshots of randomly selected web pages.

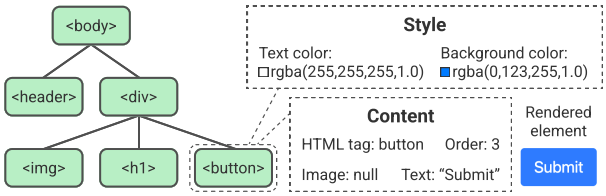

Figure 2: Data format of a web page in this study.

3. Dataset Construction for Web Page Coloring

We first describe a generic preprocessing technique that converts web pages into a tractable data format for machine learning applications. We then describe the details of the dataset we have constructed. Screenshots of randomly selected web pages from our dataset are shown in Fig. 1.

3.1. Data Format and Preprocessing

We represent a web page as an ordered tree, where each vertex is an element to be rendered on the page and has con-

tent and color style information (Fig. 2). We use HTML tags of elements, their order in siblings in the tree, and low-level features of images (*e.g.*, average color) and text (*e.g.*, word counts) as content information, please refer to the supplemental material for details. For color styles, it is not trivial to define what to extract. Also, raw web pages contain many redundant elements for training the coloring model. We discuss below our color style definition and simplification technique.

3.1.1 Color Style Definition

Style information for a web page is defined in Cascading Style Sheets (CSS), which basically consists of the specifications of target elements (*i.e.*, selector) and styles to be applied (*i.e.*, properties and their values). Here we consider two representative properties *color* and *background-color* as the color style, corresponding to the text and background colors of the element, respectively. There are two possible choices for where to obtain the values of these properties: specified or computed values.

The specified values are the values defined in the style sheets. For elements where values are not specified, browser's default CSS is applied. Available formats for the values of color properties include RGBA values, color keywords, and special keywords, such as "inherit", which causes the use of the same value as the parent element. Thus, the same color in appearance can be represented in different formats, and canonicalizing them requires extra effort. The computed values, on the other hand, are the values resolved by browser. For our target properties, the values are disambiguated to RGBA format, which can be used as the specified values without any change in appearance. We use the computed values as the canonical values for the target properties, and we additionally use the ground-truth specified values for the other properties for visualization.

3.1.2 Web Page Simplification

Raw web pages contain many elements that do not contribute to their appearance on the present screen, including alternative elements such as elements that only appear on laptop-sized screens and functional elements such as hidden input elements in submission forms. Many redundant or less relevant elements should be eliminated, as they will negatively affect the learning of the coloring model.

Here we consider keeping only those elements that contribute to the first view on the mobile screen. A naive simplification method would be to try removing an element and if the appearance of the first view does not change, then really remove that element. Removing elements, however, often breaks the original style when other elements are involved in the CSS selector such as "div $>$ a". To avoid such undesirable style corruption, before simplification, we change the CSS to a different one that uses absolute selectors such as "#element-1" and is carefully tailored to be equivalent to the original style specification. As a result, the average number of elements, excluding those placed in the head element, was significantly reduced from 1656.1 to 61.4. Note that color styles applied to a simplified web page can be applied to the original one if the correspondence of the elements is recorded before and after the simplification.

3.2. Our Dataset

The dataset should have a large enough and diverse set of web pages. We select the Klarna Product Page dataset [14] as the source of web pages, which provides mobile web pages for e-commerce in various countries and companies. Specifically, we use snapshots of the web pages provided in MHTML format, which can be replayed in a browser almost identically to the original pages. Although the procedures described in this section apply to the snapshots, they are also applicable to web pages crawled in a general way.

We implement the process of the retrieval of the computed values and the simplification of web pages using Selenium WebDriver [26] and Google Chrome [9]. Complex web pages that, even after simplification, still have more than 200 elements or a tree depth of more than 30 are excluded. We also exclude web pages encoded in anything other than utf-8 to simplify text processing. As a result, a total of 44,048 web pages are left with an average number of elements of 60.6 and an average tree depth of 9.3. The web pages of the official training split is divided 9:1 for training and validation, and those of the official test split for testing, resulting in 27,630 pages for training, 3,190 pages for validation, and 13,228 pages for testing.

4. Approach

We provide high-level explanations of our approach for the sake of clarity, and details on message passing, model architecture, and implementation are presented in the supplemental material.

4.1. Overview

We define web page colorization as the task of generating a color style for each element, given the content information and the hierarchical structure of the elements. We denote the index of an element as n, the number of elements on the page as N, the set of color style as $\mathcal{Y} = \{Y_n\}_{n=1}^{N}$, the set of content information as $\mathcal{C} = \{C_n\}_{n=1}^{N}$, and the hierarchical structure as T, respectively. Thus, the main objective of this task can be formulated as creating a function f that generates color styles based on given conditions, *i.e.*, $f : (\mathcal{C}, T) \mapsto \hat{\mathcal{Y}}$, where a variable with the hat symbol, *i.e.* $\hat{\mathcal{Y}}$, represents the estimate of that variable, *i.e.*, $\mathcal{Y}$.

Following the recent image colorization method [19], we take a two-stage approach using a core model to

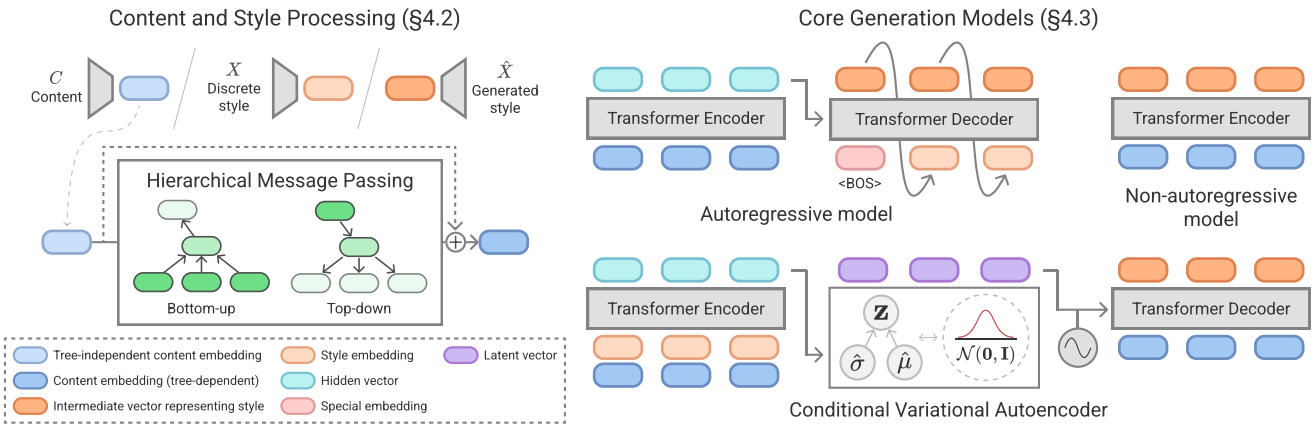

Figure 3: Key components of our method for generating discrete color styles. Our method has three variants with different core generation models, all using the same architecture of the content and style encoders and the style estimation head. Element-wise content features are contextualized by features from other elements through hierarchical message passing.

generate low-resolution discrete styles $\mathcal{X} = \{X_n\}_{n=1}^N$, *i.e.*, $g : (\mathcal{C}, T) \mapsto \hat{\mathcal{X}}$, and a color upsampler to increase the resolution of the colors to the desired size, *i.e.*, $h : (\mathcal{X}, \mathcal{C}, T) \mapsto \hat{\mathcal{Y}}$. For quantization, we divide the RGBA values into eight bins per channel, and the RGB channels are grouped together. The discrete text color of an element is then represented by a pair of $\left(x_{\text{rgb}} \in \{1, \ldots, 8^3\}, x_{\text{alpha}} \in \{1, \ldots, 8\}\right)$. The same quantization is applied to the background color. We consider three core generation models: autoregressive model, non-autoregressive model, and conditional variational autoencoder, which are all based on Transformer networks and have the same architecture for the content and style encoders and the style estimation head. The core model and the color upsampler are trained independently. Some key components of our method are shown in Fig. 3.

4.2. Content and Style Processing

The content encoder takes content information for each element $\{C_n\}_{n=1}^N$ (the index n is omitted hereafter) as input, and referring to the tree structure T, converts them into d-dimensional content embeddings $\{\mathbf{h}_C\}$ that reflect the hierarchical relationship of the elements. We implement the content encoder using bottom-up and top-down message passing with residual connections, which is expressed as:

$$\bar{\mathbf{h}}_C = \text{MaxPool}\left(\{\text{Emb}_c\left(c\right) | c \in C\}\right), \quad (1)$$

$$\{\mathbf{h}_{\text{up}}\} = \text{MP}_{\text{up}}(\{\bar{\mathbf{h}}_C\}, \mathbf{h}_{\text{leaf}}; T), \quad (2)$$

$$\{\mathbf{h}_{\text{down}}\} = \text{MP}_{\text{down}}(\{\mathbf{h}_{\text{up}}\}, \mathbf{h}_{\text{root}}; T), \quad (3)$$

$$\mathbf{h}_C = \bar{\mathbf{h}}_C + \mathbf{h}_{\text{down}}, \quad (4)$$

where $\text{MaxPool}(\cdot)$ is the max-pooling operator, $\text{Emb}_c(\cdot)$ is the embedding function corresponding to the content c, $\mathbf{h}_{\text{leaf}}$ and $\mathbf{h}_{\text{root}}$ are learnable parameters, respectively. In the bottom-up message passing MP_{up}, the value $\mathbf{h}_{\text{up}}$ of an element is computed using the value $\bar{\mathbf{h}}_C$ of the element and the values $\mathbf{h}_{\text{up}}$ of the elements children or $\mathbf{h}_{\text{leaf}}$ if the element is a leaf node. The top-down message passing MP_{down} is defined similarly in the reverse direction.

The style encoder maps a discrete color style X to a style embedding $\mathbf{h}_X \in \mathbb{R}^d$ independently for each element. Specifically, we represent an RGBA color by a vector obtained by merging two embeddings corresponding to discrete RGB and alpha values. The style embedding is obtained by merging two color vectors corresponding to text and background colors. For elements without text, we use a special learnable embedding instead of the text color vector. The style estimation head is the module in the final output part that maps intermediate vectors representing element-by-element styles to discrete RGB and alpha probabilities for text and background colors, respectively.

4.3. Core Generation Models

The role of the core generation model is to generate a discrete color style for each element $\{\hat{X}\}$ based on the given conditions. Here, we implement this model using the Transformer network and train it with two different schemes: maximum likelihood estimation (MLE) and conditional variational autoencoder (CVAE). The model learned with the MLE is further subdivided into two variants, autoregressive and non-autoregressive models, resulting in a total of three variants of the core model to be considered.

4.3.1 MLE-based Model

The models are trained by maximizing the log-likelihood. Let θ be the model parameters, the objective is written as

$$\max_{\theta} E\left[\log p_\theta\left(\mathcal{X} \,|\, \mathcal{C}, T\right)\right]. \quad (5)$$

Autoregressive Model: In the autoregressive model, the conditional probability of the color styles is modeled as

$$p_\theta\left(\mathcal{X} \mid \mathcal{C}, T\right) := \prod_{n=1}^{N} p\left(X_n \mid X_1, X_2, \cdots, X_{n-1}, \mathcal{C}, T\right). \tag{6}$$

The order of the elements is defined by the pre-order tree traversal.

Non-autoregressive Model: In the non-autoregressive model, the conditional probability is assumed to be not conditioned on the previous estimates and is modeled as

$$p_\theta\left(\mathcal{X} \mid \mathcal{C}, T\right) := \prod_{n=1}^{N} p\left(X_n \mid \mathcal{C}, T\right). \tag{7}$$

We implement these models with the Transformer networks as illustrated in the upper right of Fig. 3. The content and style embeddings and the style estimation head used are those described in Section 4.2.

4.3.2 CVAE-based Model

In general, the non-autoregressive model is capable of faster inference than the autoregressive model, but is unable to create a diverse colorizations for fixed input values. We introduce latent variables into the non-autoregressive model and extend it to the formulation of conditional variational autoencoder, allowing for the generation of diverse outputs from a single input.

Let us denote latent variables as $Z \in \mathbb{R}^{Nd}$, the conditional generative distribution as $p_\theta\left(\mathcal{X} \mid Z, \mathcal{C}, T\right)$, the posterior as $q_\phi\left(Z \mid \mathcal{X}, \mathcal{C}, T\right)$, and the prior as $p\left(Z \mid \mathcal{C}, T\right)$, respectively, the learning objective of CVAE is expressed as:

$$\begin{aligned} \max_{\phi,\theta} \quad & E_{q_\phi(Z \mid \mathcal{X}, \mathcal{C}, T)}\left[\log p_\theta\left(\mathcal{X} \mid Z, \mathcal{C}, T\right)\right] \\ & - \lambda \mathrm{KL}\left(q_\phi\left(Z \mid \mathcal{X}, \mathcal{C}, T\right) \,\|\, p\left(Z \mid \mathcal{C}, T\right)\right), \end{aligned} \tag{8}$$

where ϕ and θ are the model parameters and λ is a hyperparameter to balance the two terms. We set $\lambda = 0.1$ for all experiments.

Using multivariate Gaussian distributions with diagonal covariance matrices, we model the conditional distributions as follows:

$$q_\phi\left(Z \mid \mathcal{X}, \mathcal{C}, T\right) := \mathcal{N}\left(\mu\left(\mathcal{X}, \mathcal{C}, T\right), I\sigma\left(\mathcal{X}, \mathcal{C}, T\right)\right), \tag{9}$$

$$p_\theta\left(\mathcal{X} \mid Z, \mathcal{C}, T\right) := \prod_{n=1}^{N} p\left(X_n \mid Z, \mathcal{C}, T\right), \tag{10}$$

$$p\left(Z \mid \mathcal{C}, T\right) := \mathcal{N}\left(\mathbf{0}, I\right), \tag{11}$$

where $\mu(\cdot)$ and $\sigma(\cdot)$ are the functions that return parameters corresponding to the mean and the variance of the Gaussian distribution, respectively.

We implement this model with both the Transformer encoder and decoder as shown in the bottom right of Fig. 3. The Transformer encoder takes content and style embeddings as input and produces the estimated mean and the estimated variance for each element. They are concatenated for all elements and treated as the returned vectors of $\mu(\cdot)$ and $\sigma(\cdot)$ in Eq. (9), respectively. The latent variables Z are then sampled from the Gaussian distribution using the reparametrization trick and partitioned into a set of latent vectors equal to the number of elements. The Transformer decoder takes the latent vectors and embeddings as input and estimates the color styles of all elements. Note that the latent vectors are applied the positional encoding so that the decoder can identify which latent vector corresponds to which element.

4.4. Color Upsampler

The color upsampler takes the discrete color styles $\mathcal{X}$ as input and generates the color styles in full resolution $\hat{\mathcal{Y}}$. To force the upsampled colors to stay in their original quantization bins, we estimate the proportions in the bins instead of directly estimating the full-resolution colors. We train the Transformer-based model to minimize the mean squared error between the predicted proportions and the ground-truth proportions.

5. Experiments

We evaluate our methods against additional baseline methods in terms of the quality and diversity of the generated color styles.

5.1. Methods

We employ baseline methods based on statistics of the dataset and image colorization.

Statistics-based Coloring: We use simpler methods based on statistics of color styles. Specifically, we first collect the frequencies of the discrete color for each pair of HTML tags and CSS properties in the training set. The color is then determined by mode selection (*mode*) or frequency-weighted sampling (*sampling*) and upsampled in the same way as our methods. When sampling, we set the same color for the same pair of tags and properties, encouraging consistent color styling within a single web page. For pairs that appear in the test set but not in the training set, the global frequencies across all the tags are referenced.

Image Colorization [19]: We adapt the image colorization technique to our task as an additional baseline. The main consideration of adaptation is that both input and output are images rather than structural color information. We use a screenshot of ground-truth web page converted to a

grayscale image as the input. Obtaining structural colors from the output colorized image is not trivial, as it requires knowing the correspondences between the pixel colors and the element color styles. Inspired by chroma key compositing, we assign an unused color to a target element, render the web page, and consider the pixels with that color to be the corresponding pixels. The colorized image is then referenced and the most dominant color in the corresponding pixels is taken for the target element. We repeat this process for all the elements. Note that in this process, all colors are treated as opaque colors, *i.e.*, the alpha value is 1.0. For the specific method, we employ *ColTran* [19], a state-of-the-art method based on Transformer, whose official implementation is publicly available[1].

Our Methods: We use three methods with different core models: the autoregressive model (AR), the non-autoregressive model (NAR), and the conditional variational autoencoder (CVAE) described in Section 4. Each method uses the same color upsampler and the same hyperparameters of the Transformer network, the details can be found in the supplemental material. To generate with the AR model, we use the same trained model with three different decoding strategies: greedy decoding, top-p sampling [13] with p=0.8 and p=0.9. Note that during training, we exclude the text color of elements without text content from the loss calculation because it does not affect the appearance.

5.2. Evaluation Metrics

Accuracy and Macro F-score: As proxy metrics for the quality of the generated results, the reproducibility of the ground-truth data is measured. We compute the accuracy and the Macro F-score using the discrete RGB and alpha values. The text color of elements without text content is excluded from the computation. For the Macro F-score, the average of class-wise F-scores is used.

Fréchet Color Distance: The diversity of the generated results is another important measure. We devise a new metric, named Fréchet Color Distance (FCD), with reference to FID [12], which is widely used in the image generation and colorization tasks. In the FID, the distance between the distributions of the generated data and the real data is computed using intermediate features of the pre-trained Inception Network. These high-level image features may not represent well low-level color information, thus in the FCD, histograms of the discrete RGB colors are used as the features to compute the distribution distance. The histograms are normalized by the number of elements, and their statistics are calculated for background colors, text colors, and pixels, respectively, so that the diversity of the different perspectives can be measured. Note that the text color of elements without text content is excluded from the corresponding histogram.

Contrast violation: The quality of the generated results is evaluated from another perspective: accessibility. To investigate the accessibility of the generated results, we use Lighthouse [10], a commonly used auditing tool in practice, and consider the contrast audit, which is greatly affected by color styles. The contrast audit is based on Web Content Accessibility Guidelines 2.1 [18] and tests whether the contrast ratio between the background and text colors meets the criteria for all the text on a page. We report the percentage of pages and the average number of elements that violate the contrast audit.

5.3. Color Style Generation

We summarize the quantitative results in Table 1 and the qualitative results in Fig. 4. In the quantitative results, we report the mean and standard deviation of one evaluation for each of the three models trained with different random seeds, and for Stats (sampling), we report them for three evaluations. The qualitative results show screenshots of web pages where the generated color styles are applied. For those with multiple results, we first generate 20 variations with a single model, then the one with the largest color distance is greedily selected, starting from a random selection.

The Fréchet Color Distances (FCDs) are calculated between two halves of the test set, one from the generated results and the other from the ground-truth data. The FCDs for real data use the other ground-truth data instead of the generated results, allowing for reasonable comparisons with other methods. The real data shows a large percentage of web pages violating the contrast criterion, but this is roughly consistent with the public statistics [2]. We consider the metrics of contrast violation only for reference, and consider them good if they do not differ significantly from the real statistics.

The image colorization baseline ColTran [19] performs well for the Pixel-FCD and poorly for the other metrics, which may indicate that by using the ground-truth grayscale screenshots, ColTran can generate plausible color styles only for those with many corresponding pixels, such as the background color of the body element. In color style extraction from the colorized image, the extraction errors for those with fewer corresponding pixels, such as the color for smaller text as in the right example in Fig. 4, may be the cause of contrast violation. Due to not considering the alpha composition in the extraction, RGB values obtained differently from the ground-truth may cause poor accuracies, BG-FCD, and Text-FCD.

The statistical baselines of structural coloring show

[1] https://github.com/google-research/google-research/tree/master/coltran

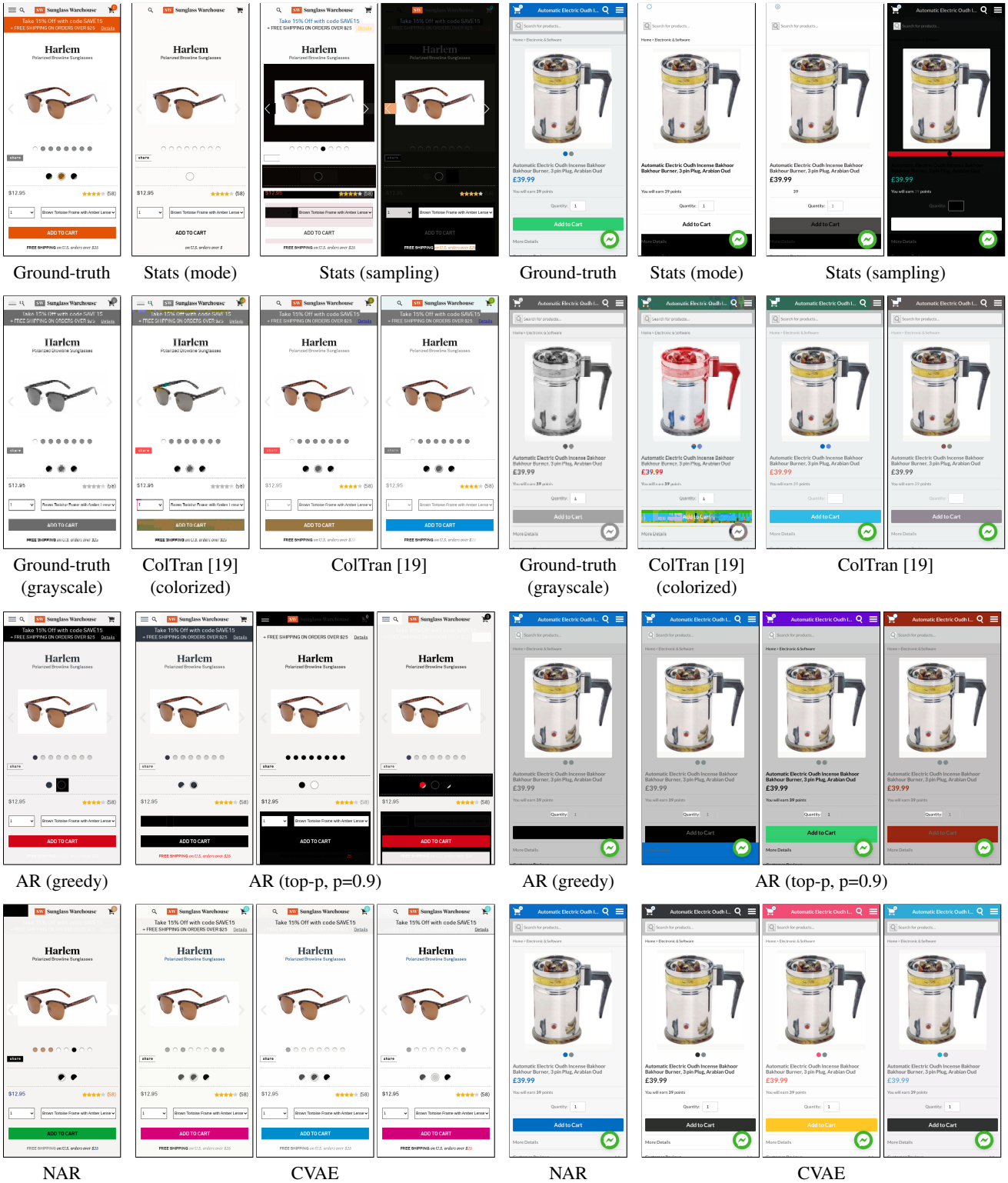

Figure 4: Qualitative comparison of color style generation. NAR and CVAE successfully generate more plausible color styles than the others, and CVAE can produce multiple variations. Best viewed in color and zoom.

Table 1: Quantitative comparison of color style generation. The methods with dagger symbols (†) generate in a deterministic manner, while the others generate in a stochastic manner. The Fréchet Color Distances are multiplied by 1e-3 for clarity. Note that only ColTran uses the ground-truth grayscale screenshot, *i.e.*, partial color information per pixel.

	Accuracy		Macro F-score		Fréchet Color Distance			Contrast violation	
Method	RGB ↑	Alpha ↑	RGB ↑	Alpha ↑	BG ↓	Text ↓	Pixel ↓	% Pages	# Elements
ColTran [19]	.285±.000	.411±.000	.009±.001	.101±.000	665.78±2.49	103.50±1.61	**3.14**±0.58	95.51±0.15	5.29±0.02
Stats (mode)†	.717	.891	.003	.219	20.94	263.11	7.43	34.17	1.18
Stats (sampling)	.621±.000	.821±.000	.004±.000	.207±.001	**0.71**±0.02	82.22±0.31	169.80±1.05	94.41±0.08	4.47±0.02
AR (greedy)†	.720±.002	.886±.002	.033±.002	.405±.003	2.93±0.33	39.19±2.61	22.98±3.21	66.43±0.70	2.02±0.02
AR (top-p, p=0.8)	.717±.003	.885±.002	.032±.002	.403±.002	2.63±0.37	33.79±3.59	33.67±0.55	71.00±1.12	2.30±0.06
AR (top-p, p=0.9)	.714±.003	.883±.003	.030±.002	.402±.006	2.40±0.33	33.00±3.81	37.44±0.30	73.42±1.01	2.40±0.06
NAR†	**.773**±.001	**.929**±.001	**.076**±.001	**.670**±.002	1.57±0.33	**21.81**±2.20	5.98±1.23	74.02±0.92	2.25±0.09
CVAE	.771±.001	**.929**±.000	.069±.001	.665±.003	1.50±0.04	28.14±1.13	4.20±0.19	74.71±0.19	2.23±0.05
CVAE w/o message passing	.762±.001	.918±.000	.062±.002	.620±.005	2.16±0.03	33.02±2.82	10.58±1.29	75.10±0.79	2.29±0.06
CVAE w/o residual connection	.768±.001	.927±.000	.064±.000	.647±.009	1.41±0.25	29.94±2.58	5.40±1.17	74.25±0.87	2.20±0.07
Real data	1.000	1.000	1.000	1.000	0.08	3.04	0.56	71.72	2.39

higher accuracies and lower macro F-scores, which indicates the imbalance of color styles where typical color styles, such as black text and white background, appear much more frequently than other color styles. The Stats (sampling) improves BG-FCD and Text-FCD and degrades Pixel-FCD compared to Stats (mode). The former is as expected, and the latter can be interpreted because typical color styles are often assigned to those with many corresponding pixels in the ground-truth data. For contrast violation, the results indicate that assigning typical color styles leads to fewer violations, while sampling background and text colors independently leads to more violations.

Our Transformer-based methods generally perform better than the other baselines, and the trends in contrast violations are similar to the real data. In AR, accuracies and FCDs show that sampling method can control the typicality of the output. Both NAR and CVAE outperform the AR variants with comparable scores and show better visual results than the others.

Ablation study: We perform an ablation study to investigate the performance contribution of the design of the hierarchical contextualization. The results using CVAE as the base model are summarized in the lower part of Table 1 and show that removing message passing degrades performance, suggesting the importance of usage of hierarchical modeling. We can also see that the performance is degraded without using the residual connection, indicating that they may help propagate only the minimum information in complex message passing.

6. Limitations and Discussion

While our methods succeed in generating relatively plausible color styles, they are still far from perfect. A typical failure case is the functional coloring that informs the product color or the current slide in a slideshow (*i.e.*, carousel), as can be found in Fig. 4. The challenge with the functional coloring stems from the fact that necessary information is removed by the simplification or is not included on the page in the first place. The necessary information is the representative color to distinguish variations for the product color, and the order of the displayed slide for the slideshow. To address this issue, explicit information needs to be added as additional content or the user needs to modify it in post-processing.

Elements with certain styles are inevitably associated with specific color styles. For example, an element with a "round" style should have a background color. However, our methods can fail to handle such styling rules. Adding color-related styles could improve the situation, but one needs to consider which and how much to add. It may be necessary to find a way to handle all the complex CSS properties in a unified and efficient manner, which may also lead to the extension of CSS property generation beyond color.

There is still room for improvement regarding the dataset. The e-commerce mobile web pages used in our study have low relevance between individual product images and color styles, making them unsuitable for evaluating the harmony between images and colors. Also, it has not been tested whether our methods can handle more elements, such as full-page view on a mobile-sized screen or on a laptop-sized screen. To facilitate these studies, we believe that the development of stable data collection tool and the construction of large-scale datasets using such a tool are promising future directions.

References

[1] Saeed Anwar, Muhammad Tahir, Chongyi Li, Ajmal Mian, Fahad Shahbaz Khan, and Abdul Wahab Muzaffar. Image Colorization: A Survey and Dataset. *arXiv:2008.10774v3*, 2022.

[2] The HTTP Archive. HTTP Archive: Accessibility. `https://httparchive.org/reports/accessibility#a11yColorContrast`. (accessed 2022-08-27).

[3] Tony Beltramelli. Pix2code: Generating code from a graphical user interface screenshot. In *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS'18, 2018. doi: 10.1145/3220134.3220135.

[4] Murtuza Bohra and Vineet Gandhi. ColorArt: Suggesting colorizations for graphic arts using optimal color-graph matching. In *Proceedings of Graphics Interface*, GI'20, 2020. doi: 10.20380/GI2020.11.

[5] Huiwen Chang, Ohad Fried, Yiming Liu, Stephen DiVerdi, and Adam Finkelstein. Palette-based photo recoloring. *ACM Transactions on Graphics*, 34(4), 2015. doi: 10.1145/2766978.

[6] Xingyu Chen, Zihan Zhao, Lu Chen, JiaBao Ji, Danyang Zhang, Ao Luo, Yuxuan Xiong, and Kai Yu. WebSRC: A Dataset for Web-Based Structural Reading Comprehension. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, EMNLP'21, 2021. doi: 10.18653/v1/2021.emnlp-main.343.

[7] Dianne Cyr, Milena Head, and Hector Larios. Colour appeal in website design within and across cultures: A multi-method evaluation. *International Journal of Human-Computer Studies*, 68(1), 2010. doi: 10.1016/j.ijhcs.2009.08.005.

[8] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for Quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning*, ICML'17, 2017.

[9] Google. Google Chrome - Download the Fast, Secure Browser from Google. `https://www.google.com/chrome/`. (accessed 2022-08-29).

[10] Google. Lighthouse. `https://github.com/GoogleChrome/lighthouse`, 2022. (accessed 2022-08-29).

[11] Zhenyu Gu and Jian Lou. Data driven webpage color design. *Computer-Aided Design*, 77, 2016. doi: 10.1016/j.cad.2016.03.001.

[12] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs trained by a two time-scale update rule converge to a local nash equilibrium. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NeurIPS'17, 2017.

[13] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. In *International Conference on Learning Representations*, ICLR'20, 2020.

[14] Alexandra Hotti, Riccardo Sven Risuleo, Stefan Magureanu, Aref Moradi, and Jens Lagergren. The Klarna Product Page Dataset: A Realistic Benchmark for Web Representation Learning. *arXiv:2111.02168v2*, 2021.

[15] Naz Kaya and Helen H. Epps. Relationship between color and emotion: A study of college students. *College Student Journal*, 38(3), 2004.

[16] Kotaro Kikuchi, Mayu Otani, Kota Yamaguchi, and Edgar Simo-Serra. Modeling Visual Containment for Web Page Layout Optimization. *Computer Graphics Forum*, 40(7), 2021. doi: 10.1111/cgf.14399.

[17] Suzi Kim and Sunghee Choi. Dynamic closest color warping to sort and compare palettes. *ACM Transactions on Graphics*, 40(4), 2021. doi: 10.1145/3450626.3459776.

[18] Andrew Kirkpatrick, Joshue O Connor, Alastair Campbell, and Michael Cooper. Web Content Accessibility Guidelines (WCAG) 2.1. `https://www.w3.org/TR/2018/REC-WCAG21-20180605/`, 2018. (accessed 2022-05-14).

[19] Manoj Kumar, Dirk Weissenborn, and Nal Kalchbrenner. Colorization transformer. In *International Conference on Learning Representations*, ICLR'21, 2021.

[20] Ranjitha Kumar, Arvind Satyanarayan, Cesar Torres, Maxine Lim, Salman Ahmad, Scott R. Klemmer, and Jerry O. Talton. Webzeitgeist: Design mining the web. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI'13, 2013. doi: 10.1145/2470654.2466420.

[21] Sharon Lin, Daniel Ritchie, Matthew Fisher, and Pat Hanrahan. Probabilistic color-by-numbers: Suggesting pattern colorizations using factor graphs. *ACM Transactions on Graphics*, 32(4), 2013. doi: 10.1145/2461912.2461988.

[22] Peter O'Donovan, Aseem Agarwala, and Aaron Hertzmann. Color compatibility from large datasets. *ACM Transactions on Graphics*, 30(4), 2011. doi: 10.1145/2010324.1964958.

[23] Arnold Overwijk, Chenyan Xiong, and Jamie Callan. ClueWeb22: 10 Billion Web Documents with Rich Information. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR'22, 2022. doi: 10.1145/3477495.3536321.

[24] Gabriel Peyré and Marco Cuturi. Computational Optimal Transport: With Applications to Data Science. *Foundations and Trends in Machine Learning*, 11(5-6), 2019. doi: 10.1561/2200000073.

[25] Qianru Qiu, Mayu Otani, and Yuki Iwazaki. An Intelligent Color Recommendation Tool for Landing Page Design. In *27th International Conference on Intelligent User Interfaces*, IUI'22 Companion, 2022. doi: 10.1145/3490100.3516450.

[26] Selenium. WebDriver — Selenium. `https://www.selenium.dev/documentation/webdriver/`. (accessed 2022-08-29).

[27] Gapsy Studio. How to Choose UI Colors for Mobile and Web Design Wisely. `https://gapsystudio.com/blog/ui-design-colors/`, 2020. (accessed 2022-05-17).

[28] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NeurIPS'17, 2017.

[29] Polina Volkova, Soheila Abrishami, and Piyush Kumar. Automatic Web Page Coloring. In *Advances in Visual Computing*, Lecture Notes in Computer Science, 2016. doi: 10.1007/978-3-319-50835-1_9.

[30] Nanxuan Zhao, Ying Cao, and Rynson W.H. Lau. Modeling fonts in context: Font prediction on web designs. *Computer Graphics Forum*, 37(7), 2018. doi: 10.1111/cgf.13576.

[31] Tianming Zhao, Chunyang Chen, Yuanning Liu, and Xiaodong Zhu. GUIGAN: Learning to Generate GUI Designs Using Generative Adversarial Networks. In *Proceedings of the 43rd International Conference on Software Engineering*, ICSE'21, 2021. doi: 10.1109/ICSE43902.2021.00074.