GENERATING UI CODE FROM SCREENSHOTS OF WEBSITES
GROUP 7

Srinitya Kondapally
skonda29@asu.edu

Chsallet Christina Kancharla
ckancha1@asu.edu

Leepaakshi Gokulkrishnan
lgokulkr@asu.edu

Pooja Laxmi Shanmugananthan
pshanmu6@asu.edu

1 Abstract

User interface (UI) design is evolving at a rapid pace, which underscored the need for effective ways to convert visual designs into useful code. Using a machine learning-driven framework, this project bridges the gap between UI design and implementation by automatically creating responsive HTML code from GUI screenshots. By utilizing a deep learning encoder-decoder architecture, the technique generates structured HTML code based on Bootstrap using Recurrent Neural Networks (RNNs) with Long Short-Term Memory (LSTM) units and extracts visual features from GUI designs using Convolutional Neural Networks (CNNs). The code is refined for deployment by post-processing, while a coupled parser model guarantees syntactic correctness. Through process simplification, improved design-to-code coherence, and error reduction, this methodology seeks to transform the software development lifecycle, offering a scalable and adaptable solution for diverse UI designs.

2 Introduction

In software development, the gap between UI design and implementation remains a persistent challenge, with manual translation of designs into HTML and CSS consuming significant time and resources. This process is prone to inconsistencies and errors, especially as the complexity of web and mobile interfaces continues to grow.

To address this, the project introduces a machine learning-driven solution that automates the conversion of static UI designs into functional, responsive code. Utilizing convolutional neural networks (CNNs) to analyze visual elements and recurrent neural networks (RNNs) for structured code generation, this approach ensures design fidelity while dramatically reducing development time and errors.

The motivation behind this innovation lies in its potential to streamline workflows, enhance collaboration between designers and developers, and free up resources for more strategic tasks. By addressing a critical bottleneck in the software development lifecycle, this project offers a transformative step toward greater efficiency, consistency, and productivity in modern development practices.

3 Project Description

This project focuses on developing a machine learning framework that automates the generation of responsive HTML code from graphical user interface (GUI) screenshots. The goal is to streamline the traditionally manual and error-prone process of converting visual designs into structured web code by leveraging an encoder-decoder architecture.

The methodology employs a modular design based on deep learning techniques. The encoder-decoder architecture comprises two main components. First, a convolution neural network (CNN) serves as the encoder, extracting feature representations from GUI screenshots that capture the layout and design elements. Second, a recurrent neural network (RNN) with Long-Short-Term Memory Units (LSTM) functions as the decoder, interpreting the extracted features and generating the corresponding Bootstrap code. The outputs of these two components are integrated via a parser model, which maps visual features to their corresponding code elements while ensuring syntactic correctness. Finally, a post-processing step converts the generated Bootstrap code into functional HTML, preparing it for deployment.

This approach emphasizes modularity and adaptability, enabling the encoder-decoder architecture to accommodate diverse design styles and platforms. By addressing the technical challenges of GUI-to-code translation, this project lays a foundation for enhancing automation in software development, with the potential to extend its applicability to mobile and desktop applications.

4 Related Work

Automated code generation has seen significant advancements in recent years, with several deep learning models designed to translate various types of input into executable code. Pix2Code is one of the pioneering works in GUI-to-code generation, where it used a convolutional neural network (CNN) to extract features from GUI screenshots and a recurrent neural network (RNN) to generate intermediary code. [1] This approach demonstrated the potential for bridging the gap between visual design and code, but it struggled with scalability and accuracy across diverse and complex datasets.

Building on the foundations established by Pix2Code, subsequent research has explored various enhancements, particularly by integrating more sophisticated architectures such as transformers and attention mechanisms. These models, often used in natural language based source code generation, are better at capturing long-range dependencies and can be adapted to generate code from a broader range of input descriptions. However, while these advancements have improved the performance of code generation in textual contexts, challenges remain in effectively handling visual data and ensuring accurate mapping between GUI elements and corresponding code [2] [3]. The present level of design-to-code automation has been studied by [4], identified major challenges that include managing intricate user interface layouts and incorporating contextual awareness into code generation models. More recently, new methods for improving GUI-to-code translation have also been studied. The use of convolutional neural networks with class activation mapping to create web pages from mockup designs was suggested in [5] and resulted in notable accuracy gains.

Building on these foundations, this project improves upon pix2code by incorporating advanced architectures, such as BiLSTMs and GRUs, and refining outputs through post-processing steps to ensure syntactic correctness. Additionally, we introduce a post-processing step that compiles generated intermediary code into functional HTML, ensuring design fidelity and reducing errors. By combining these innovations, our model is able to handle more complex GUI designs and provides a scalable solution for automated code generation. This work contributes to the ongoing effort to refine and extend Pix2Code, offering a more robust and flexible framework for generating code from visual design inputs. The study underscores recent developments in the use of attention-based architectures and big datasets to attain state-of-the-art performance in GUI-to-code systems, as investigated by [6].

5 Dataset

The dataset used in this study is a subset of the Pix2Code dataset, which consists of pairs of GUI screenshots and corresponding code for multiple platforms, including web-based technologies, Android, and iOS. The dataset is publicly available and has been widely used for training models in tasks related to GUI-to-code generation. For this study, web-based samples were used, containing 1,749 pairs of GUI screenshots and code. These pairs serve as the training and evaluation set, with 350 samples from each platform allocated to the evaluation set, while the remaining samples were used for training. The table 1 summarizes the key aspects of the pix2code dataset.

6 Approach

This study employs an encoder-decoder architecture designed to automate the generation of HTML code from GUI screenshots, which bridges the gap between visual design and structured web code. The architecture consists of three main components: a vision model, a language model, and a combined parser, as shown in 1, working together to generate high-quality, syntactically correct HTML code. The encoder-decoder setup enables the model to process and transform complex visual input into sequential code output.

Vision Model (Encoder)
The first component, the vision model, serves as the encoder in this architecture. It utilizes a Image Feature Extraction Architecture such as Convolutional Neural Network (CNN) or Vision Transformer (ViT) to process the GUI screenshot and extract its visual features. They are particularly effective for image processing tasks due to their ability to capture

Table 1: Key Features of the Pix2Code Dataset

Aspect	**Details**
Dataset Name	pix2code
Platforms	iOS, Android, Web
Dataset Composition	pairs of GUI screenshots and corresponding code
Input type	GUI Screenshots
Total Samples	1,749 pairs of GUI screenshots and code (web-based)
Training set	1,399 samples
Evalation set	350 samples
Output type	Generated code (domain-specific language)

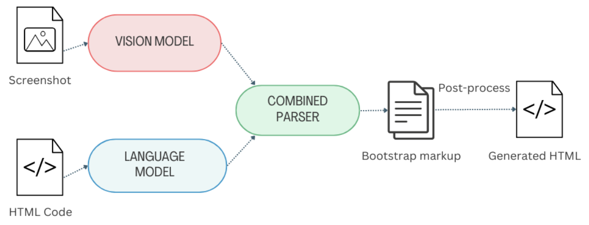

Figure 1: Model flow architecture

spatial hierarchies in visual data. This part of the framework is tasked with identifying key elements such as buttons, containers, headers, and other UI components from the raw pixel data of the screenshot. This feature extraction process transforms the image into a feature vector, which serves as the input for the next stage of the pipeline.

Language Model (Decoder)
The second component is the language model, which functions as the decoder in the encoder-decoder architecture. This model is based on Recurrent Neural Networks (RNNs), specifically Long Short-Term Memory (LSTM) units and Gated Recurrent Units (GRU), which are ideal for modeling sequential data such as code [7]. The LSTM and GRU models process the feature vector from the image classification architecture (CNN/ViT) and generate a sequence of tokens that corresponds to the HTML code. These RNN-based models learn to map the extracted visual features to relevant code elements, such as <div>, <button>, <span>, and other HTML tags. LSTMs and GRUs are particularly effective in this context because they can capture long-range dependencies in sequences, which is crucial for maintaining the correct structure of the code.

Combined Parser
The third key component of the model is the combined parser, which integrates the outputs of the vision and language models. The parser maps the visual features encoded by the CNN to the code generated by the LSTM, ensuring that the final output is both syntactically and semantically correct. This step is crucial for ensuring that the generated code accurately reflects the layout and functionality of the original design. The parser operates in a supervised learning framework, allowing it to learn the mappings between visual features and code elements during the training phase. The parser model is designed to ensure that the HTML code generated follows the hierarchical structure of the GUI elements.

Post-Processing (Compilation to HTML)
Once the intermediate code, typically in Bootstrap format, is generated by the language model, it undergoes a post-processing step. This step involves converting the Bootstrap code into fully functional HTML code. The post-processing algorithm ensures that the output is not only syntactically correct but also functionally valid, allowing for integration

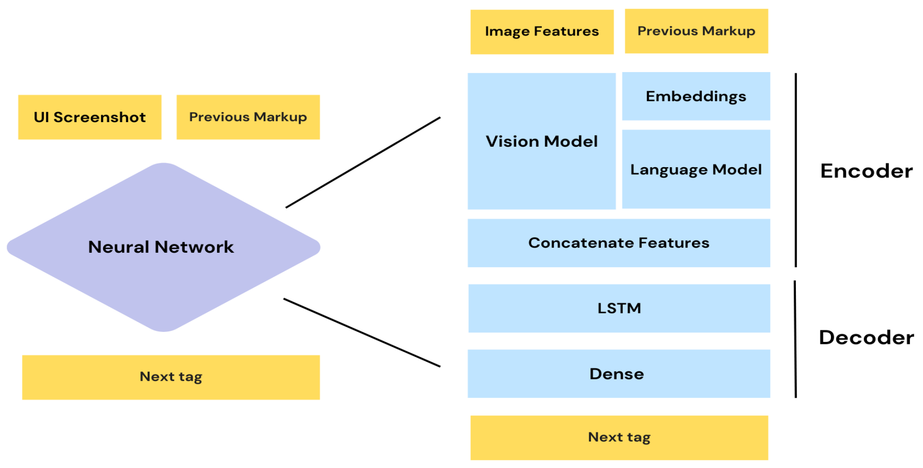

Figure 2: Encoder-Decoder model Architecture for UI Code Generation

into live applications. This step is essential for validating the generated code and ensuring it reflects the original design's layout and functionality. The complete architecture describe above is shown in the figure 2.

7 Methods Experimented

This section outlines the experimental methods employed in our code generation task. The approach involves two primary components: Vision models for processing UI screenshots and Language models for code sequence analysis.

7.1 Vision Model

For the task of generating code from a UI screenshot, we experimented with several established image architectures to extract meaningful feature representations from the UI images. These feature vectors are then used for downstream code generation via a combined LSTM parser. This section evaluates DenseNet-121, VGG-16, ResNet-50, and ViT-Base/16 with respect to their effectiveness in feature extraction for UI-to-code mapping.

Dense (Fully Connected Layers)
In our experiments, we employed fully connected dense layers to extract features from the image representations. These layers, which consist of multiple neurons fully connected to the previous layer, are particularly effective for creating compact and high-dimensional representations from visual data. The dense layers serve to consolidate the information from the previous layers, distilling important patterns in the UI structure. This approach was useful in capturing the key features of UI components, particularly for models focused on understanding relationships between different elements in a UI layout before passing them to the LSTM parser for code generation [8].

VGGNet
VGG-16 [9], a well-known architecture with 16 layers, was used for its simplicity and consistent feature extraction. The model's straightforward design, which utilizes 3×3 convolutional filters, ensures reliable representation of UI elements such as text fields, images, and buttons. Although computationally expensive, VGG-16 provides clear and interpretable feature vectors that are particularly useful in generating structured code, as it captures hierarchical patterns commonly found in UI layouts. Its max-pooling layers help in reducing spatial dimensions, making it effective in representing compact UI components.

ResNet
ResNet-50, a residual network consisting of 50 layers, was selected for its ability to handle deeper architectures through the use of residual connections [10]. These connections allow the network to learn more complex representations without suffering from vanishing gradients. In the context of UI-to-code generation, ResNet-50's depth enables the model to extract high-level features and fine-grained details, making it well-suited for detecting intricate UI layouts and understanding spatial relationships between UI components. The bottleneck design with 1×1, 3×3, and 1×1 convolutions improves efficiency without sacrificing performance.

ViT
The ViT-Base/16 architecture was utilized for its ability to capture global context through the self-attention mechanism of the transformer model [11]. By dividing the input image into 16×16 patches, ViT processes the entire UI image as a sequence of patches, allowing it to model long-range dependencies across the UI layout. This capability is particularly beneficial for generating code from UIs with complex layouts, where understanding the overall structure and alignment of UI components is crucial. ViT-Base/16's ability to produce high-quality embeddings from such global features makes it a powerful choice for UI-to-code generation tasks.

7.2 Language Model

In this section, we discuss three encoder architectures—LSTM, GRU, and BiLSTM—that are utilized for feature extraction from code sequences in the UI-to-code generation pipeline.

LSTM
Long Short-Term Memory (LSTM) networks are designed to capture long-range dependencies in sequential data [12]. LSTMs process code tokens and learn hierarchical relationships between them. They are effective in modeling sequential code patterns, typically consisting of 2-3 layers, each with 256 or 512 hidden units. The architecture efficiently captures context and order within code sequences.

GRU
Gated Recurrent Units (GRU) are a more computationally efficient alternative to LSTMs, designed to capture long-term dependencies with fewer parameters. GRUs are simpler and faster to train while maintaining the ability to model sequential dependencies in code. Like LSTMs, they are typically structured with 2-3 layers and 256 or 512 hidden units but offer a more lightweight solution[13].

BiLSTM
Bidirectional LSTM (BiLSTM) networks process input sequences in both forward and backward directions, allowing the model to capture contextual dependencies from both past and future tokens [14]. This bidirectional approach enhances the model's ability to understand complex code structures. BiLSTMs are typically composed of 2-3 layers, with 256 or 512 hidden units, and are well-suited for tasks requiring comprehensive context understanding.

8 Training

The training of the models involved experimenting with different methods for the image and language models. The image models were trained to extract visual features from GUI screenshots, while the language models were trained to generate Bootstarp code sequences from these extracted features. Both the image and language models were optimized using a standard cross-entropy loss function, which is commonly used for sequence generation tasks. The training utilized the Adam optimizer with an initial learning rate of 1e-3, which was adjusted over time. A batch size of 32 was used, and dropout rates were applied to reduce overfitting. The models were trained for several epochs, with early stopping to prevent overtraining. Hyperparameters, such as learning rate and dropout rates, were tuned based on the performance on a validation set, ensuring optimal convergence and reliable model performance.

9 Evaluation and Performance metrics

BLEU Score
The BLEU score (Bilingual Evaluation Understudy) is a widely used metric for evaluating the quality of machine-generated sequences, particularly in tasks like machine translation and code generation. The BLEU score with 4-grams measures the overlap of n-grams (in this case, 4-grams) between the predicted and reference sequences, with higher values indicating better alignment. This metric provides a quantitative measure of the accuracy and fluency of the generated output by comparing it to a set of reference sequences.

Pixel to Pixel accuracy
Pixel-to-pixel accuracy is a key performance metric in evaluating the model's ability to reconstruct a user interface from a screenshot. Measures the proportion of pixels in the predicted UI layout that correspond to the ground truth. High pixel-to-pixel accuracy signifies that the model has effectively captured the visual details and spatial relationships of UI elements, such as buttons, text fields, and images. This precision is vital for generating accurate and func-

tional code that mirrors the original UI design, ensuring the generated code's UI elements align seamlessly with the input.

While qualitative evaluation of the model was straightforward by comparing the similarities between the ground-truth HTML code and the generated output, we also explored several quantitative metrics to measure model performance. These included categorical cross-entropy loss, BLEU score, perplexity, and pixel-by-pixel comparison between the ground truth and the generated image. Although pixel-by-pixel comparison offered valuable insight into visual accuracy, we opted to prioritize the BLEU score due to its computational efficiency and its stronger focus on assessing the quality of the generated HTML code. Among the various metrics, the 4-gram BLEU score proved to be the most effective in evaluating the model's ability to produce accurate and coherent HTML code.

10 Results

The table 2 presents a comparative analysis of the performance of various model architectures across two key evaluation metrics - BLEU-Score and Pixel-to-Pixel accuracy. Among the models tested, the DenseNet-BiLSTM combination

Table 2: Model Performance: BLEU-Score and Pixel-to-Pixel Accuracy

Model	**BLEU-Score (%)**	**Pixel-to-Pixel Accuracy (%)**
FCL - LSTM	84.25	80.50
FCL - GRU	85.30	81.15
VGGNet - LSTM	86.00	92.00
VGG16Net - GRU	87.10	93.70
VGG16Net - BiLSTM	87.50	94.20
ResNet - LSTM	90.00	90.00
ResNet - GRU	91.00	91.00
ResNet - BiLSTM	92.00	91.50
ViT - GRU	90.00	85.10
ViT - LSTM	90.10	85.20
ViT - BiLSTM	91.00	85.80
DenseNet - LSTM	94.00	94.00
DenseNet - GRU	95.25	95.10
DenseNet - BiLSTM	**97.35**	**96.00**

outperformed all other models, achieving the highest BLEU-Score of 97.35% and the highest Pixel-to-Pixel Accuracy of 96.00%. This model demonstrated a significant improvement in both metrics compared to other models, particularly in BLEU-Score, where it surpassed the next best model, DenseNet-GRU (95.25%), by over 2 percentage points. Additionally, the DenseNet-BiLSTM model also led in Pixel-to-Pixel Accuracy, outperforming DenseNet-GRU by 0.90% points.

ResNet-based models, specifically ResNet-BiLSTM, demonstrated strong performance, with a BLEU-Score of 92.00% and Pixel-to-Pixel Accuracy of 91.50%, showing their ability to balance high-quality text generation and pixel accuracy. However, they did not surpass DenseNet models in either metric.

The ViT models, while exhibiting reasonable performance, showed a relatively lower Pixel-to-Pixel Accuracy compared to other architectures, with the best result being a BLEU-Score of 91.00% and Pixel-to-Pixel Accuracy of 85.80% achieved by ViT-BiLSTM.

In summary, the DenseNet-BiLSTM model not only performed the best overall but also exhibited a notable edge in both BLEU-Score and Pixel-to-Pixel Accuracy, highlighting its potential as the most robust architecture in this study for generating high-quality outputs.

11 Contribution

Chsallet Christina Kancharla - Contributed to dataset preparation. Helped in image vector mapping and feature vector mapping. Contributed to the literature review on related work in code generation. Worked on the milestone report. Explored VGGNet (VGG-16) and LSTM, GRU architecture.

Leepaakshi Gokulkrishnan - Drafted the project proposal. Contributed to dataset preparation. Helped in code vector mapping. Reviewed the milestone report. Performed analysis and comparison. Explored DenseNet and all language model architectures. Performed analysis and comparison for the project. Worked on the final report.

Pooja Laxmi Shanmugananthan - Performed data preprocessing. Helped in image vector mapping. Reviewed the milestone report. Performed analysis and comparison. Explore VGGNet, Vision Transformer (ViT) and all language model architectures. Performed analysis and comparison for the project . Worked on the final report.

Srinitya Kondapally - Literature survey about the existing code generation models and their architectures. Performed data preprocessing. Helped in code vector mapping and feature vector mapping. Worked on milestone report. Explored ResNet (ResNet-50) and all language model architectures. Worked on the final report.

12 Conclusion and Future Work

This project successfully demonstrated the feasibility of automating UI-to-code translation using deep learning. The DenseNet-BiLSTM architecture achieved state-of-the-art performance, highlighting the potential of this approach in reducing development time and enhancing accuracy.

The project adhered closely to the proposed milestones. Early-stage tasks such as dataset acquisition and preprocessing were completed on schedule, setting a strong foundation for model development. Subsequent phases, including model experimentation with architectures like CNN-GRU and BiLSTM, progressed as planned, with evaluations conducted using BLEU and Pixel-to-Pixel metrics. The results align with expectations, with DenseNet-BiLSTM outperforming other configurations, confirming the success of the chosen approach.

Future directions include integrating Large Language Models (LLMs) like GPT to improve context understanding, extending the system to support modern front-end frameworks such as React and Vue.js, incorporating attention mechanisms to better capture complex relationships in UI layouts, and expanding the training dataset to include diverse and complex GUI designs for improved generalizability.

References

[1] T. Beltramelli. pix2code: Generating code from a graphical user interface screenshot. *arXiv preprint arXiv:1705.07962*, 2017.

[2] Å. Stige, E. D. Zamani, P. Mikalef, and Y. Zhu. Artificial intelligence (ai) for user experience (ux) design: a systematic literature review and future research agenda. *Information Technology People*, 2023.

[3] T. Kaluarachchi and M. Wickramasinghe. A systematic literature review on automatic website generation. *Journal of Computer Languages*, 75:101202, 2023.

[4] C. Si, Y. Zhang, Z. Yang, R. Liu, and D. Yang. Design2code: How far are we from automating front-end engineering? *arXiv preprint arXiv:2403.03163*, 2024.

[5] A. A. J. Cizotto, R. C. T. de Souza, V. C. Mariani, and L. dos Santos Coelho. Web pages from mockup design based on convolutional neural network and class activation mapping. *Multimedia Tools and Applications*, 82(25):38771–38797, 2023.

[6] H. Laurençon, L. Tronchon, and V. Sanh. Unlocking the conversion of web screenshots into html code with the websight dataset. *arXiv preprint arXiv:2403.09029*, 2024.

[7] Tomas Mikolov, Martin Karafiát, Luká Burget, Jan Honza ernocký, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Interspeech*, 2010.

[8] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks, 2018.

[9] Srikanth Tammina. Transfer learning using vgg-16 with deep convolutional neural network for classifying images. *International Journal of Scientific and Research Publications (IJSRP)*, 2019.

[10] Hoa Khanh Dam, Truyen Tran, and Trang Pham. A deep language model for software code, 2016.

[11] Yixing Xu, Chao Li, Dong Li, Xiao Sheng, Fan Jiang, Lu Tian, and Ashish Sirasao. Fdvit: Improve the hierarchical architecture of vision transformer. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 5927–5937, 2023.

[12] Baidya Nath Saha and Apurbalal Senapati. Long short term memory (lstm) based deep learning for sentiment analysis of english and spanish data. In *2020 International Conference on Computational Performance Evaluation (ComPE)*, pages 442–446, 2020.

[13] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling, 2014.

[14] Ezat Ahmadzadeh, Hyunil Kim, Ongee Jeong, Namki Kim, and Inkyu Moon. A deep bidirectional lstm-gru network model for automated ciphertext classification. *IEEE Access*, 10:3228–3237, 2022.