From Mock-Ups to IFML-like GUI Models: Using Large Language Models in Web Engineering

Atefeh Nirumand[1] and Jordi Cabot[1,2]

[1] Luxembourg Institute of Science and Technology, Luxembourg
{atefeh.nirumand, jordi.cabot}@list.lu
[2] University of Luxembourg, Luxembourg

Abstract. The development of web applications often requires the translation of design mock-ups into functional code, a process that is time-consuming and inaccessible to those without technical expertise. This research addresses this challenge by leveraging Large Language Models (LLMs) to automate the derivation of IFML-like Graphical User Interface (GUI) models from mock-up images. These models serve as a foundation for the validation and subsequent generation of web applications using available deterministic code-generators. By enhancing the BESSER low-code framework with this extension, we provide an advanced pipeline that integrates LLMs into model-driven web application development relying on the UML (for the structural models) and IFML (for the GUI ones) standards. Our LLM-based approach achieves high accuracy for both digital and hand-drawn mock-ups, demonstrates efficiency with rapid run-time performance, and delivers web pages that have a high structural similarity to their corresponding mock-ups. The complete infrastructure is available in an open-source repository.

Keywords: Large Language Models (LLMs) · IFML · Graphical User Interface (GUI) · Design-to-Code · Web Engineering.

1 Introduction

Front-end development for software systems, including web applications, focuses on creating visually appealing and responsive GUIs. In the early stages, mock-ups are often used to clarify design intentions and validate these designs with stakeholders. Indeed, these mock-ups, ranging from low-fidelity wireframes to high-fidelity visuals, help ensure accurate implementation of UI elements [19,2]. Tools like Figma allow mock-up creation and front-end code export (e.g., HTML) but lack flexibility for full-stack development, including back-end logic and data management. While technologies such as HTML, CSS, and JavaScript have evolved and become more user-friendly, they still require technical expertise to employ them to code front-ends, which is a challenge for individuals and small businesses lacking technical skills [22,10]. Additionally, the need for a back-end to handle data flow and logic further increases development complexity. Manual coding

is error-prone, and template-based methods often lack the flexibility to address unique business needs [10].

Model-Driven Development (MDD) has emerged as a solution to the increasing complexity of software development, including web application development. This is achieved by utilizing standardized modelling languages such as the Interaction Flow Modelling Language (IFML) [16] to abstract UI design and functionality. IFML-based models visually represent UIs along with their navigation flows and interactions with business logic, enabling a structured, reusable framework. This reduces errors, automates repetitive tasks, improves development efficiency, and enhances flexibility. Moreover, low-code development, a style of MDD [4], simplifies the development workflows by combining visual design tools and pre-built generator/interpreter components, reducing the need for manual coding.

The research [22,10,11,20] identifies key areas for enhancing automatic website generation: converting screen captures into code, utilizing LLMs, creating multi-page websites, and enhancing scalability through back-end integration. Addressing these challenges can further democratize web development, enabling users of all skill levels to build complex applications. However, current solutions often prioritize direct sketch to front-end code generation via AI-based coding assistants, neglecting intermediate modeling representations such as IFML.

Neglecting intermediate modeling representations, such as IFML, can lead to several drawbacks, including lack of consistency, difficulty in maintaining and validating the code, and limited flexibility for scaling across different platforms. Using IFML helps address these problems. From the IFML models, deterministic code generators (or model interpreters) could derive the final application. In contrast to non-deterministic direct sketch-to-code approaches, in our approach, the final code does not need to be re-verified.

In this sense, our proposal aims to bring the best of both worlds: AI and MDD. We use AI for the sketch-to-modeling phase and MDD for the models-to-code one. For the AI phase, we rely on the last generation of LLMs and use prompt engineering [5,13] to maximize the accuracy of the results for this specific task. We have evaluated the approach using 100 case studies from the CodePen[3] repository, focusing on accuracy, flexibility, and run-time performance. In addition, we have integrated a Django-based CRUD application generator into the workflow. This generator translates extracted GUI models into functional web applications, adhering to Django best practices and supporting seamless front-end and back-end integration. Thanks to the generator, we can provide a full AI-enhanced model-driven pipeline for the generation of web applications from an initial set of mock-ups.

The rest of the paper is organized as follows: Section 2 details the proposed approach. Section 3 describes the tool support, while Section 4 presents the evaluation. Section 5 discusses limitations and threats. Section 6 reviews related work, and finally, Section 7 concludes the paper and highlights potential areas for future research.

[3] https://codepen.io/

2 Approach Overview

This section presents an LLM-driven, low-code approach to automate the extraction of UML class diagrams and IFML-like GUI models from mock-up images. The versatility of LLMs enables processing diverse data inputs, making our approach adept at efficiently handling various UI mock-up contexts. The work focuses on two key aspects:

1. **Structural Model Extraction**. Utilizing LLM to derive Structural models (also known as domain models or data models) in the form of UML class diagrams from UI mock-ups. A Structural model is needed to derive functional GUIs capable of performing CRUD operations (Create, Read, Update, and Delete) on data (typically stored in a relational database). Instead of asking the designer to provide this model, we aim to infer it from the mock-up itself.
2. **GUI Model Extraction**: Utilizing LLM to produce IFML-like GUI models that: 1) depict the mock-up GUI structure and 2) are aligned with the Structural model obtained in the preceding step.

This is depicted in Figure 1. As shown in the picture, once the models have been inferred, a Django [6] code generator uses them to create a fully functional CRUD web application. An extension for complex, multi-page applications is discussed in Section 2.3.

Note that our approach has been implemented on top of the BESSER low-code platform [1]. BESSER comes with a B-UML (standing for BESSER Universal Modeling Language) language with packages to represent UML class diagrams and IFML-like GUI models. BESSER is python-based and therefore Jinja is chosen as a template-engine for creating and executing the code generators. Technical details are available in Section 3.

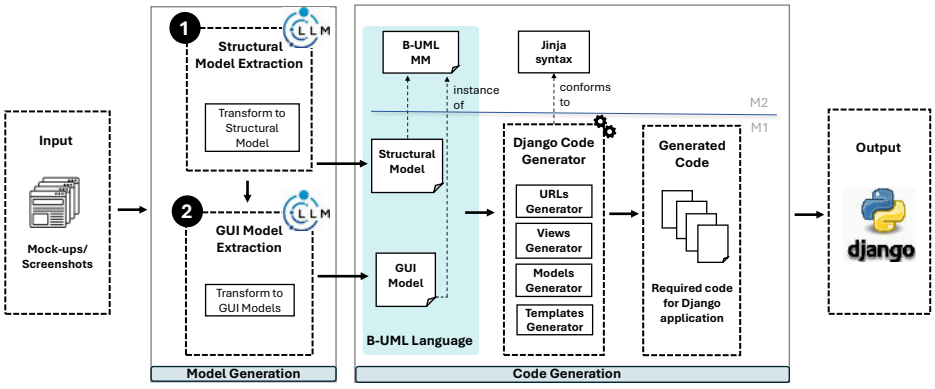


Fig. 1: Proposed Approach: Transforming UI Mock-Ups (Hand-Drawn or Digital) into IFML-like GUI Models with an Integrated Django Code Generation.

2.1 Structural Model Extraction

To extract Structural model, we ask the LLM to generate a PlantUML[4] description from the UI mock-ups passed as images to the LLM. As shown in Figure 1, this method supports screenshots as input, extending applicability to web page captures. We have chosen PlantUML as it is a textual syntax, which is easier for LLMs to generate. Among the available textual syntaxes, PlantUML is assumed to be one of the most frequently encountered in LLM training data due to its popularity. Additionally, Conrardy and Cabot [7] have demonstrated that LLMs are able to provide correct PlantUML syntax from images.

To generate PlantUML descriptions, a task-specific prompt guides the LLM to ensure an accurate and valid generation of PlantUML text. As demonstrated in Listing 2.1, this prompt specifies how the UI elements identified by the LLM are translated into structured representations, effectively mapping each UI image to a corresponding PlantUML class. Multi-shot prompting was considered, but one-shot prompting proved highly effective and simpler. This process ensures alignment between the generated structural representations and the application design. An alternative is to have the designer provide the Structural model as a separate input, but this way we simplify the development flow.

The prompt outlines valid attribute types (*int*, *float*, *str*, *bool*, *time*, *date*, *datetime*, *timedelta*) aligned with BESSER's supported data types for mapping PlantUML to the Structural model. These types are assigned based on UI data components. Adherence to PlantUML syntax prevents errors like duplicate associations or invalid methods. Class relationships mirror logical UI connections, ensuring consistency and avoiding redundancy. The approach covers essential components (classes, attributes, relationships), and PlantUML descriptions map directly to BESSER's internal representation, available as an import option. Given a hand-drawn mock-up of a simplified library app with three pages (Figure 2), the LLM generates a UML class diagram based on key entities like books, libraries, and authors. For brevity, we focus on the *Book* page. The structural model for this page is shown in Listing 2.2.

```
Task: Generate a PlantUML code that represents the Structural model of an application
      based on the provided UI image from one page of the application.

Description: You will receive a UI image that represents one page of a web
     application. Your goal is to analyze the provided UI image and use it to
     generate a PlantUML model that accurately represents the application's structure
      in a PlantUML file. Please note that each UI image corresponds to one class in
     PlantUML, representing a single page of the web app.
Instructions:
     1. You will be given an example UI image and its corresponding PlantUML code,
          demonstrating how each UI component maps to the application's structural
          model.
     2. Based on the provided UI image (and the example as guidance), generate a
          PlantUML file that represents the web page's structure.
     3. Ensure the generated PlantUML does not have any syntax errors.
     4. Make sure to consider the following attribute types: int, float, str, bool,
          time, date, datetime, and timedelta in resulting PlantUML code.
     5. Ensure that the PlantUML code does not contain methods within the class
          definition.
     6. Please ensure that buttons are not treated as attributes for any classes in
          the PlantUML code.
     7. Ensure the generated PlantUML includes all relevant classes, attributes, and
          relationships between components, accurately reflecting the entire
          application's design as seen in the UI image.
```

Listing 2.1: Direct Prompt for Extracting PlantUML code from UI Mock-up.

[4] https://plantuml.com/

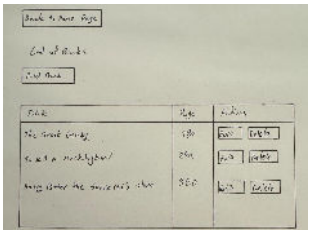


Fig. 2: UI Mock-Up for the *Book* Page in the Library Example.

```
# Classes
Book = Class(name="Book") # Other info omitted for brevity

# Book class attributes and methods
Book_pages: Property = Property(name="pages", type=IntegerType)
Book_title: Property = Property(name="title", type=StringType)
Book.attributes = {Book_pages, Book_title}  # Other info omitted for brevity

# Relationships
writes: BinaryAssociation = BinaryAssociation(
    name="writes",
    ends={Property(name="Author", type=Author, multiplicity=Multiplicity(1, 9999)),
          Property(name="Book", type=Book, multiplicity=Multiplicity(1, 1))
    }) # Other info omitted for brevity

# Domain Model
domain_model = DomainModel(
    name="Domain Model",
    types={Library, Author, Book},  # Other info omitted for brevity
)
```

Listing 2.2: Code excerpt of the Structural Model.

2.2 GUI Model Extraction

After extracting the Structural model, the next phase focuses on deriving IFML-like GUI models from the images of UI mock-ups. Each mock-up is analyzed to generate a GUI model instantiating a simplified IFML metamodel.

Figure 3 presents a simplified excerpt of this metamodel, focusing on the GUI elements central to our analysis. The green metaclasses represent the structural metamodel of B-UML. The complete metamodel is available in our repository[5]. Due to space limitations, a detailed explanation is omitted; for a comprehensive discussion, refer to our previous work [15].

Note that, as BESSER is in Python, the models follow the syntax of IFML-like GUI metamodel, serving as textual representations of the GUI model. These differ from plain Python code, like the Django code generated later.

A two-step methodology was employed for the GUI model extraction: Direct Prompting and Self-Improvement Prompting. As shown in Listing 2.3, in **Direct Prompting** step, the LLM generates the GUI model elements corresponding to the mock-up, ensuring alignment with both the Structural model and the IFML-like metamodel.

[5] https://besser.readthedocs.io/en/latest/buml_language/model_types/gui.html

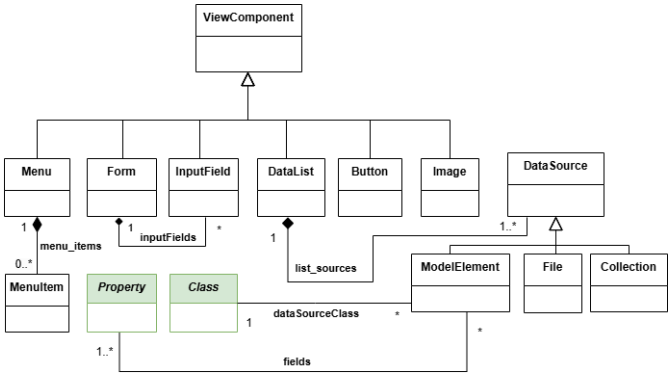


Fig. 3: Subset of the IFML-like GUI Metamodel.

```
Task: Generate Python code that accurately represents the GUI elements of a given UI
      mock-up and ensures full compliance with the defined GUI metamodel.

Description: The UI mock-up provided corresponds to a page of a web app. The entities
       of the app, their attributes, and relationships are defined within a structural
       model represented by Python code. This mock-up corresponds to one of the
      classes within the structural model. In this page, you will consider all or a
      subset of the class attributes. You can refer to the class attributes as per the
       structural model, incorporating them into the Python code generation.

Instructions:
      1. I will provide you with example UI mock-ups and their corresponding Python
           code outputs.
      2. I will also provide you with the UI Mock-up as input.
      3. You will also receive an image of the GUI metamodel along with a description
           file to guide the code generation.
      4. Your goal is to generate Python code that accurately represents all GUI
           elements from the mock-up provided.
      5. The generated Python code must resemble the structure of the provided code
           samples, while the content should be derived from the received UI mock-up.
      6. It is essential that the generated code adheres to the GUI metamodel.
      7. Ensure that you incorporate the examples provided and align the generated
           code with the expected structure and formatting.
      8. Once ready, you will be presented with a UI mock-up image, and your task is
           to generate Python code representing the GUI elements in the image.
```

Listing 2.3: Direct Prompt for Extracting GUI Model from UI Mock-up.

The second step, the **Self-improvement Prompting** process is an iterative process, refining the generated code by comparing it to the mock-up and correcting inconsistencies. This process ensures alignment with the Structural model and removes unnecessary metaclasses. The number of iterations is predefined or determined dynamically based on thresholds such as achieving accurate mock-up reflection and eliminating redundant metamodel definitions. A fail-safe mechanism triggers if no improvement is detected for a fixed number of iterations or when the maximum iteration limit is reached. Listing 2.4 shows an excerpt of the GUI model for the *Book* page, including data sources, buttons, lists, and screens. The *BookListScreen* features a back button, an add button, and a book list, while the application *MyApp* organizes these elements within *MyModule*. Detailed instructions are available in our repository[6].

```
# DataSource for Book
datasource_book: ModelElement = ModelElement(
```

[6] https://github.com/BESSER-PEARL/BESSER

```
    name="Book Data Source",
    dataSourceClass=Book,
    fields=[Book_title, Book_pages]
)

# Button for adding a book
addBookButton: Button = Button(...)  # Other info omitted for brevity

# Button for navigating back to the home page
backButton: Button = Button(...)  # Other info omitted for brevity

# List for displaying books with edit and delete actions
bookList: DataList = DataList(...)  # Other info omitted for brevity

# Screen for displaying the list of books
BookListScreen: Screen = Screen(
    view_elements={backButton, addBookButton, bookList}  # Other info omitted for
        brevity
)

# Module definition
MyModule: Module = Module(
    screens={BookListScreen}  # Other info omitted for brevity
)

# Application definition
MyApp: Application = Application(
    name="Library Management",
    modules={MyModule}  # Other info omitted for brevity
)
```

Listing 2.4: Code excerpt of the GUI Model (illustrating the *Book* page).

2.3 Multi-Page Web App Generation

So far, we have dealt with each page individually. This section proposes an extension that, given some additional navigational information in the mock-up, it manages to consolidate individual GUI models in a unified multi-page model representation.

The extension is based on a series of prompts that: 1) create a unified and consistent global model with all GUI models generated before, 2) refine the result, and 3) use additional graphical information from the mock-up to derive a navigation layout among the pages in the unified model.

In what follows, we detail each prompt. As shown in Listing 2.5, the **Direct Prompt** enables integration into a unified structure, ensuring consistent navigation, adherence to the GUI metamodel, and organizing screens within a single Python module for easy access and extensibility.

```
Task: Generate Python code that represents the integration of GUI elements from
     multiple pages of a web application into a single GUI model.

Description: The GUI Python files that you will receive are related to different
     pages of a web application. Each Python file represents a specific page's GUI
     elements. This structure is defined in the structural model by Python code,
     representing the entities, their attributes, and the relationships between them.
      Your goal is to integrate these individual GUI models into a cohesive Python
     file that represents the entire web application's UI design elements. Ensure
     that the integration maintains consistency in design and functionality across
     the different pages.

Instructions:
      1. You will be provided with example GUI models for each page of the web
           application and its expected Python output for the whole of the
           application.
      2. You will also provide the GUI Python files that are related to different
           pages of a web application.
      3. Your task is to combine these individual GUI models into a single Python
           code representation that reflects the complete UI design of the
           application.
      4. The integrated code should ensure smooth navigation between the different
           pages and maintain a coherent user experience.
      5. The Python code should adhere to the GUI metamodel provided for the web
           application.
```

```
6. Make sure to follow the design principles and layout specifications of each
    page while integrating them into the final GUI model.
7. Consider one module for the app that has a 'screens' attribute including all
    screens of the app, similar to the style of the provided example.
```

Listing 2.5: Direct Prompt for integrating GUI models into a unified model.

To improve integration, we implemented a **Self-improvement Prompt** to refine the generated Python code. Similar to the process used for individual pages, this prompt ensures alignment between the combined GUI model and individual GUI models by guiding the removal of redundant elements and correcting inconsistencies. Detailed instructions are available in our repository[7].

Finally, to manage the navigation logic between web pages, a **Navigation Logic Prompt** was developed. As shown in Listing 2.6, this prompt guides the LLM to generate navigation flows using two inputs: 1) a plain text navigation order file specifying the screen sequence and 2) a simple image illustrating the navigation flow, where connections between screens are visually defined. This ensures accurate, consistent navigation logic in the generated code.

```
Please review the Python code below. You will receive a text file that specifies the
    order of pages. Please reorder the content based on the order specified in a
    text file, but in the reverse order compared to the provided sequence. You will
    receive an image illustrating the navigation flow between multiple screens of
    the web application. Complete the code by just specifying target screens for
    buttons with 'actionType=ButtonActionType.Navigate'. Focus solely on adding the
    target screen attribute for buttons with 'actionType=ButtonActionType.Navigate'.
     Once you've made the necessary additions, kindly respond with the updated
    version of the Python code.
```

Listing 2.6: Navigation Logic Prompt.

By combining these three prompts, direct integration, self-improvement, and navigation logic, our approach streamlines the generation of a complete GUI model from individual page representations, ensuring robustness and consistency in the resulting GUI model. The unified GUI model, along with the Structural model, was then used to generate Django code for the entire application, covering all three pages of the library example. The generated *Book* page, adhering to Django's default styling, is shown in Figure 4.

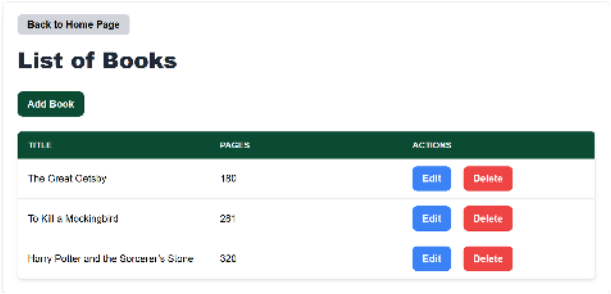


Fig. 4: Generated Web Page for the *Book* Page in the Library Example.

[7] https://github.com/BESSER-PEARL/BESSER

In the process of designing our prompts, we tested multiple styles (e.g., zero-shot, multi-shot, step-by-step) and selected the one that produced the most coherent outputs, strongest alignment with the LLM's training data, and most reliable performance. Our strategy incorporates best practices from OpenAI's prompt engineering guidelines[8], including role-based prompts, assistant-style examples, and structured output formatting.

To evaluate the fidelity of the generated web pages, we used an image similarity algorithm based on OpenAI's CLIP model. Both input and generated UI images were converted to grayscale, binarized to remove styling, and then encoded using CLIP's ViT-B/32 model. Cosine similarity between feature vectors quantified layout alignment. The model prioritizes structural layout over visual details like color, font, or size, aligning evaluation with our goal of generating functional, CRUD-optimized web pages rather than aesthetic replicas. Results showed 86% similarity for the *Library* page, 72% for the *Author* page, and 87% for the *Book* page, demonstrating the effectiveness of our approach. A more systematic evaluation, based on examples from the CodePen repository, is described in Section 4.

3 Tool Support

The full approach is implemented on top of the BESSER [1] platform, integrating MDD and AI for automating front-end and back-end generation with an IFML-like metamodel for GUI modeling. We use GPT-4o for prompt execution due to its high accuracy and ability to handle complex tasks like GUI modeling and back-end integration [17]. While other LLMs can be used, GPT-4o has shown the best performance in our tests, providing reliable and precise results.

Moreover, BESSER has also been enhanced with a Django code generator (with additional details available in our repository[9]) that converts GUI and Structural models into functional web components, completing the full pipeline presented in this paper. The generator handles both back-end and front-end tasks, creating a database schema using Django's ORM for the back-end and building a responsive, CRUD GUI for the front-end. A similar approach could be adapted to other ecosystems.

4 Evaluation

We evaluate the effectiveness of the proposed approach for UI mock-ups into IFML-like GUI models, by answering the following research questions:

RQ1 (Accuracy): To what extent does the generated GUI model accurately represent the original mock-up?

[8] https://platform.openai.com/docs/guides/prompt-engineering/six-strategies-for-getting-better-results#six-strategies-for-getting-better-results

[9] https://besser.readthedocs.io/en/latest/generators.html

RQ2 (Flexibility): Can the approach handle different UI mock-ups with different styles and complexities?
RQ3 (Run-time Performance): How efficient is the approach in processing mock-ups and generating GUI model?

To evaluate the approach, the CodePen repository was selected for its diverse real-world web pages. A total of 100 case studies (categorized in Section 4.2) were manually selected across 13 UI categories, ensuring a representative sample. Examples with overly simple layouts (e.g., plain text, no interactivity) were excluded, with preference given to pages featuring diverse UI elements (e.g., buttons, inputs) to reflect realistic interface complexity. Our experiments were conducted on a device with Windows 11 Enterprise, an Intel(R) Core(TM) i7-1280P 1.80 GHz CPU, and 32.0 GB RAM.

Table 1: Accuracy Results of the approach applied to Digital UI mock-ups.

Feature (widget(s))	**#GT**	**Approach**		
		TP	FN	FP
Items	39	39	0	3
Menu	8	8	0	2
Button	223	213	10	2
Input Field	371	361	10	1
Form	59	59	0	0
Data List	42	41	1	0
Property*	144	144	0	0
Image	10	7	3	0
Total	896	872	24	8
Precision P = TP /(FP + TP)		**99%**		
Recall R = TP / (FN + TP)		**97%**		
F-measure = 2pr/(p+r)		**98%**		
Legend: GT: Ground Truth, TP: True positive, FP: False positive, FN: False negative				

* This element pertains to the fields within each Data List.

4.1 Accuracy (RQ1)

The approach supports both digital and hand-drawn mock-ups. Accuracy was assessed through precision, recall, and F-measure metrics, focusing on structural aspects of UI elements rather than layout details (e.g., colors, sizes).

Digital UI Mock-ups. A total of 100 CodePen case studies, each with a corresponding HTML file and UI image, were analyzed, achieving 99% precision, 92% recall, and 98% F-measure (see Table 1). The evaluation process involved manually reviewing each case study and comparing, for each one, the *ground truth* consisting of the UI elements in the original HTML files, with the UI model elements in the UI model extracted from the corresponding image. This

analysis identified True Positives (correctly identified elements), False Positives (inaccurately extracted elements), and False Negatives (missed elements).

Hand-Drawn UI Mock-ups. For hand-drawn UI mock-ups, we created equivalent hand-drawn versions for 15 case studies selected from 13 categories (listed in Table 3) on CodePen. The same evaluation approach was applied to these images, and the results, presented in Table 2, demonstrate precision (98 %), recall (99%), and F-measure (98%).

These results demonstrated the efficacy of the approach in accurately extracting GUI models from both digital and hand-drawn mock-ups. To ensure replicability, we provide a package with 100 CodePen examples, their generated GUI models, and data from the hand-drawn evaluation[10].

Table 2: Accuracy results for approach applied to Hand-Drawn UI mock-ups.

Feature (widget(s))	**#GT**	**Approach**		
		TP	FN	FP
Menu	0	0	0	1
Button	13	13	0	0
Input Field	75	74	1	0
Form	12	11	1	0
Data List	3	3	0	0
Property*	9	9	0	0
Total	112	110	2	1
Precision P = TP /(FP + TP)		**98%**		
Recall R = TP / (FN + TP)		**99%**		
F-measure = 2pr/(p+r)		**98%**		
Legend: GT: Ground Truth, TP: True positive, FP: False positive, FN: False negative				

* This element pertains to the fields within each Data List.

4.2 Flexibility (RQ2)

The flexibility is assessed indirectly through precision measurement across 13 UI mock-up categories, reflecting various interaction types, complexity levels. These categories, created manually using relevant keywords from the CodePen search engine (since CodePen lacks categorization), encompass both application domains and distinct UI components. See the categories in Table 3. The quality results (see RQ1) were consistent across all categories.

4.3 Run-time Performance (RQ3)

The run-time performance of the approach was evaluated by measuring execution times for the two key phases: phase 1, Structural model extraction, which

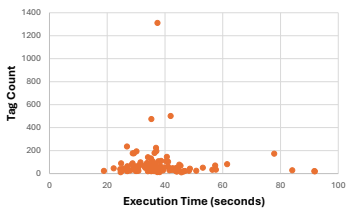
[10] https://github.com/AtefehNirumandJazi/MockupsToGUI_Replicability/tree/main

Table 3: Categories of UI Mock-ups.

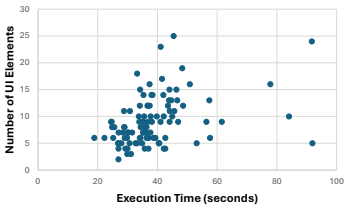
Row	Category	#Apps	Row	Category	#Apps
1	Social	6	8	Pet	7
2	Educational	19	9	Library	1
3	Media	3	10	E-commerce	3
4	Contact Manager	7	11	CRUD	17
5	Registration	9	12	Dashboard	5
6	Productivity	17	13	Authentication	2
7	Inventory	4	**Total Apps**		**100**

averaged 8.88 seconds, and phase 2, GUI model extraction, which averaged 27.85 seconds. The total average run-time was 36.96 seconds, reflecting the approach's efficiency in extracting GUI models from UI images.

As shown in Figure 5, we analyzed the relationship between application size (measured by tag and UI elements) and processing time using 100 case studies. While larger models tend to take longer, the relationship is not strictly linear. For all case studies, processing time remains below 100 seconds, with a median of 36.52 seconds and an interquartile range of 9.89 seconds, indicating the method's efficiency and consistency.



(a) Processing Time vs. Tag Count



(b) Processing Time vs. Number of UI elements

Fig. 5: Scatter Plots of Processing Time vs. Tag Count and Number of UI elements: Evaluating across 100 Case Studies in the evaluation.

5 Limitations and Threats to Validity

Limitations. While the proposed approach exhibits strong accuracy and flexibility in handling diverse UI mock-ups, it has notable limitations. It focuses primarily on identifying UI elements and does not account for design choices such as colors, sizes, text fonts, or other aesthetic features, which are crucial for user experience. Additionally, it does not address precise positional accuracy, spatial alignment, or hierarchical relationships between elements, leaving room for further refinement.

Threats to Validity. The reliability of the evaluation is contingent upon the quality of the dataset and the nuances of the implementation, with the results being influenced by the diversity of the datasets utilized. To address this, we employed CodePen, a well-recognized repository of diverse web mock-ups. A generalization issue persists, as the findings may not be applicable to unconventional designs or novel interaction paradigms. To mitigate this, we evaluated categories like social apps, e-commerce, productivity tools, educational apps, and CRUD applications, reflecting globally common web functionalities[11].

6 Related Work

This section reviews related studies pertinent to our research, grouped into different categories.

Model-Driven Approaches. Rivero et al. [18] pioneered a model-driven framework that translates mock-ups into UI models. Building on this work, Sinha and Karim [21] extended the paradigm by compiling mock-ups into flexible interfaces for various platforms. However, both approaches emphasize adaptability but lack focus on multi-page navigation and offer limited results by not relying on more powerful LLM-driven strategies as we do.

Visual-to-Code Translation. Huang et al. [9] use Convolutional Neural Networks (CNNs) for visual-to-code translation, generating static layouts from visual inputs. Pix2code [3] combines CNNs and Long Short-Term Memory (LSTM) networks for visual-to-code translation, handling image processing and code generation. Liu et al. [14] have improved contextual accuracy using bidirectional sequential models. However, these methods do not address back-end integration or multi-page navigation nor use software models as intermediate representations. Kim et al. [12] and Yan et al. [24] employed deep learning algorithms for the identification of components in GUI images. They could serve as alternatives to LLM-driven strategies and be integrated into our model-driven pipeline if the primary LLM-based method fails for niche mock-ups. CNN-based approaches may be effective in specialized domains where training datasets are available.

Leveraging LLMs for Model Extraction. Conrardy and Cabot [7] demonstrate the versatility of LLMs in extracting UML-like models from hand-drawn class diagrams. In contrast, our work derives UML class diagrams and IFML-like GUI models from UI mock-ups for web application design.

Front-End Code Generation. Image2emmet [23] uses template matching and heuristic rules, while Ferreira et al. [8] applies machine learning for front-end code generation. Both lack back-end integration and a model-based approach. Similarly, Wan et al. [22] and Design2Code [20] highlight limitations in end-to-end design-to-code systems. Our work addresses these gaps in full-stack automation.

While existing methods focus on GUI model extraction and front-end code generation, our approach integrates structural models, GUI models, and naviga-

[11] https://www.similarweb.com/top-websites/

tion logic for a deterministic model-to-code generation once the models are automatically created leveragin LLMs. We believe this combination offers a better trade-off towards the semi-automatic development of maintainable web applications. Moreover, by leveraging a model-based approach, we can generate code for multiple platforms.

7 Conclusion and Future Work

This work proposes a novel approach, based on a combination of LLMs and MDD, to speed up the development of web applications. The proposed approach offers a scalable, low-code solution that streamlines workflows, reduces costs, and democratizes web development for users of varying expertise.

Future work includes refining prompt strategies for better alignment with different LLMs, particularly for multi-page applications. We plan to integrate complementary image-to-model techniques (e.g., for deployment models) and add a conversational interface for iterative refinement of prototypes. Additionally, we aim to explore how mock-up characteristics (e.g., clarity, fidelity) affect extraction accuracy and introduce additional evaluation metrics to assess the quality and similarity between mock-ups and generated models.

Acknowledgments. This research is supported by the Luxembourg National Research Fund (FNR) through the PEARL program under grant agreement 16544475.

References

1. Alfonso, I., Conrardy, A., Sulejmani, A., Nirumand, A., Ul Haq, F., Gomez-Vazquez, M., Sottet, J.S., Cabot, J.: Building besser: an open-source low-code platform. In: International Conference on Business Process Modeling, Development and Support. pp. 203–212. Springer (2024)
2. Bajammal, M., Mazinanian, D., Mesbah, A.: Generating reusable web components from mockups. In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering. pp. 601–611. ACM (2018)
3. Beltramelli, T.: pix2code: Generating code from a graphical user interface screenshot. In: Proceedings of the ACM SIGCHI symposium on engineering interactive computing systems. pp. 1–6 (2018)
4. Cabot, J.: Positioning of the low-code movement within the field of model-driven engineering. In: Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings. pp. 1–3 (2020)
5. Chang, Y., Wang, X., Wang, J., Wu, Y., Yang, L., Zhu, K., Chen, H., Yi, X., Wang, C., Wang, Y., et al.: A survey on evaluation of large language models. ACM Transactions on Intelligent Systems and Technology **15**(3), 1–45 (2024)
6. Chen, S., Ahmmed, S., Lal, K., Deming, C.: Django web development framework: Powering the modern web. American Journal of Trade and Policy **7**(3), 99–106 (2020)
7. Conrardy, A., Cabot, J.: From image to uml: First results of image based uml diagram generation using llms. arXiv preprint arXiv:2404.11376 (2024)

8. Ferreira, J.S., Restivo, A., Ferreira, H.S.: Automatically generating websites from hand-drawn mockups. In: Proceedings of the 16th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (2021)
9. Huang, R., Long, Y., Chen, X.: Automaticly generating web page from a mockup. In: SEKE. pp. 589–594 (2016)
10. Kaluarachchi, T., Wickramasinghe, M.: A systematic literature review on automatic website generation. Journal of Computer Languages **75**, 101202 (2023)
11. Kaluarachchi, T., Wickramasinghe, M.: Webdraw: A machine learning-driven tool for automatic website prototyping. Science of Computer Programming **233**, 103056 (2024)
12. Kim, B., Park, S., Won, T., Heo, J., Kim, B.: Deep-learning based web ui automatic programming. In: Proceedings of the 2018 Conference on Research in Adaptive and Convergent Systems. pp. 64–65 (2018)
13. Liu, P., Yuan, W., Fu, J., Jiang, Z., Hayashi, H., Neubig, G.: Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. ACM Computing Surveys **55**(9), 1–35 (2023)
14. Liu, Y., Hu, Q., Shu, K.: Improving pix2code based bi-directional lstm. In: 2018 IEEE International Conference on Automation, Electronics and Electrical Engineering (AUTEEE). pp. 220–223. IEEE (2018)
15. Nirumand Jazi, A., Alfonso, I., Cabot, J.: Low-code flutter application development solution. In: Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems. pp. 838–847 (2024)
16. OMG: O.m.g.: About interaction flow modeling language specification (2025), https://www.omg.org/spec/IFML
17. Openai: Gpt-4o mini: advancing-efficient-intelligence (2025), https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/
18. Rivero, J.M., Rossi, G., Grigera, J., Burella, J., Luna, E.R., Gordillo, S.: From mockups to user interface models: an extensible model driven approach. In: Current Trends in Web Engineering: 10th International Conference on Web Engineering ICWE 2010 Workshops, Vienna, Austria, July 2010, Revised Selected Papers 10. pp. 13–24. Springer (2010)
19. Samir, M., Elsayed, A., Marie, M.I.: A model for automatic code generation from high fidelity graphical user interface mockups using deep learning techniques. International Journal of Advanced Computer Science & Applications **15**(3) (2024)
20. Si, C., Zhang, Y., Yang, Z., Liu, R., Yang, D.: Design2code: How far are we from automating front-end engineering? arXiv preprint arXiv:2403.03163 (2024)
21. Sinha, N., Karim, R.: Compiling mockups to flexible uis. In: Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering. pp. 312–322 (2013)
22. Wan, Y., Wang, C., Dong, Y., Wang, W., Li, S., Huo, Y., Lyu, M.R.: Automatically generating ui code from screenshot: A divide-and-conquer-based approach. arXiv preprint arXiv:2406.16386 (2024)
23. Xu, Y., Bo, L., Sun, X., Li, B., Jiang, J., Zhou, W.: image2emmet: Automatic code generation from web user interface image. Journal of Software: Evolution and Process **33**(8), e2369 (2021)
24. Yun, Y.S., Jung, J., Eun, S., So, S.S., Heo, J.: Detection of gui elements on sketch images using object detector based on deep neural networks. In: Proceedings of the Sixth International Conference on Green and Human Information Technology: ICGHIT 2018. pp. 86–90. Springer (2019)