Politecnico di Torino

ScuDo
Scuola di Dottorato ~ Doctoral School
WHAT YOU ARE, TAKES YOU FAR

Doctoral Dissertation
Doctoral Program in Artificial Intelligence (37^{th}cycle)

From Human Representations to AI Realization: Algorithms and Tools for Creating and Refining Interactive Systems

By

Tommaso Calò

******

Supervisor(s):
Prof. Luigi De Russis, Supervisor
Prof. Fulvio Corno, Co-Supervisor

Politecnico di Torino
2025

Declaration

I hereby declare that, the contents and organization of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

Tommaso Calò
2025

* This dissertation is presented in partial fulfillment of the requirements for **Ph.D. degree** in the Graduate School of Politecnico di Torino (ScuDo).

Abstract

This dissertation investigates novel methods and tools that employ AI to address the gaps between domain expertise and technical implementation, allowing both **domain professionals** and **end users** to effectively create and customize interactive systems.

Domain professionals face distinct obstacles based on their expertise areas. Web designers, for instance, struggle with the translation gap between visual concepts and functional implementation, leading to compromised designs, extended development cycles, and frustrating miscommunications between design and development teams. We address these challenges by developing design-to-code translation systems that automatically convert visual mockups into functional code while allowing designers to explore stylistic elements and control interactive behaviors.

Educators encounter a distinct set of barriers when creating digital learning environments. Despite possessing deep pedagogical knowledge, they typically lack the technical skills to implement effective instructional interfaces without developer assistance. We introduce a generative AI-based system that transforms teaching requirements directly into interactive interfaces, maintaining educator control over educational design while automating technical implementation.

For end users, we focus on individuals seeking to create websites which face a fundamental barrier in the programming knowledge required for development. They must either invest significant time learning technical skills or accept the creative limitations of template-based solutions. We propose a language interface creation algorithm that enable conversational website design and modification without coding expertise, and a tool for customization to user preferences.

Lastly, this dissertation focuses on smart home inhabitants and the encountered friction when their environments fail to understand natural communication patterns, forcing them to adapt to rigid command structures rather than systems adapting to

users, we introduce a multimodal disambiguation system that combine visual and textual cues to clarify user intentions, creating more intuitive interaction experiences.

Through user studies and technical evaluations across web interfaces, intelligent tutoring systems, and smart home environments, results show that designing AI collaboration systems which allow humans to effectively express their intentions and aligning the AI realization with these expressions, leads to significant improvements in user satisfaction, efficiency, and output quality compared to traditional systems.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context

According to the European Commission's Digital Economy and Society Index [1], 90% of jobs across all sectors require at least some level of digital skills, indicating widespread computer use in the workplace. While most people have now become familiar with basic computer functionality and interfaces, developing new applications or modifying existing ones to effectively support user goals still demands programming expertise that remains beyond the reach of the average user. In fact, computer programmers constitute about 0.96% of the total workforce [2].

Enhancing user participation in the initial design of systems addresses only part of the challenge, as user requirements remain diversified, constantly evolving, and often difficult to precisely identify. A more comprehensive approach is enabling *users themselves to continuously adapt the systems to their needs*. The research area pursuing this goal is called End-User Development, defined as "*a set of methods, techniques, and tools that allow users of software systems, who are acting as non-professional software developers, at some point to create, modify, or extend a software artifact.*" [3] Traditional EUD systems relied on techniques such as programming by example (also called programming by demonstration), visual programming, macros, and scripting languages. Recently, the rapid emergence of generative AI technologies is creating a new horizon for empowering end users to actively create and adapt interactive systems. These new model can produce executable code and user-facing behaviors from naturalistic human descriptions.

Rather than relying on formal syntax or domain-specific notations, users can now issue prompts in everyday language (e.g., "Generate a custom web-based quiz with adaptive questions based on user performance") and obtain functional code stubs or fully formed applications. This amplification of end-user agency draws on a longstanding EUD theme: *lowering or altogether removing the barriers imposed by traditional programming environments*.

This shift, sometimes referred to as the "generative turn" in end-user development [4], may encourage faster experimentation with software features and richer adaptation of interactive systems.

Compared to conventional methods in end-user development, generative AI offers a lower threshold but also requires new forms of guidance. Where previous end-user development approaches focused on macros, scripting environments, and visual programming, generative AI extends these possibilities by letting users fluidly express their goals and instructions in terms familiar to them, which are then translated into software artifacts. However, end users are only providing prompts and passively accepting whatever result emerges, they may confront situations where subtle errors, incomplete requirements, or security vulnerabilities become difficult to diagnose. These implications underscore the importance of *hybrid design strategies* that weave together the fluid expressiveness of generative AI with the predictable scaffolding of more conventional EUD methods such as visual layouts, parameter menus, or constrained domain notations. In this way, generative AI technologies complement the EUD vision by promising more ambitious, highly customized applications that better capture the user's changing needs.

We can identify two types of end-user activities from a user-centered design perspective:

1. **Parameterization or customization.** Activities that allow users to choose among alternative behaviors (or presentations or interaction mechanisms) already available in the application. Adaptive systems are those where the customization happens automatically by the system in reaction to observation the user's behavior.

2. **Program creation and modification.** Activities that imply some modification, aiming at creating from scratch or modifying an existing software

artifact. Examples of these approaches are: programming by example, visual programming, macros, and scripting languages.

This dissertation investigates novel AI techniques to bridge the gap between users' intentions and technical implementation. Specifically, it focuses on enabling both creation and customization capabilities across three domains: web interfaces, intelligent tutoring systems, and smart home environments. For web interfaces, the research targets professional designers needing to rapidly prototype visual concepts and end users wanting to create custom websites. In intelligent tutoring systems, it addresses educators' needs to develop effective learning environments. For smart homes, the work focuses on enabling everyday users to personalize their spaces through intuitive interactions.

These works span a period (*circa* 2021–2025) of extraordinary change in machine–learning capabilities. The earliest investigations (Chapters 2 and 3) were carried out when heuristic pipelines and recurrent models such as LSTMs still constituted the most reliable option for UI translation or domain reasoning. Beginning in 2022, the public availability of large transformer-based models (e.g. the GPT family) unlocked entirely new possibilities for interface generation and natural-language reasoning. The later chapters (4 and 5) therefore build directly on these large language models, while Chapter 6 reflects on the broader implications of this transition. Where relevant, each chapter explicitly acknowledges the technological constraints that shaped the chosen methods at the time of experimentation, thereby highlighting the progressive incorporation of generative AI throughout the thesis.

1.2 Background

Creating and customizing interactive systems remains challenging for many users, despite advances in digital technologies. Significant technical barriers persist, particularly among domain experts without programming backgrounds and everyday users seeking to personalize their digital experiences. These barriers limit the democratization of digital tools and prevent many potential users from creating solutions tailored to their specific needs. This dissertation investigates three application areas where these barriers emerge most prominently: (i) web development, where professional designers face a translation gap between visual concepts and functional implementa-

tion and end users struggle with the technical complexity of website creation and customization; (ii) intelligent tutoring systems, where educators struggle to transform their pedagogical knowledge into effective digital learning environments; and (iii) smart home technologies, where everyday users encounter rigid interaction models that fail to accommodate natural communication patterns.

1.2.1 Accelerating and Democratizing Web Development

The process of creating and customizing web interfaces presents distinct challenges for professional designers in their workflow and end users wanting to build web systems for their businesses or personal needs.

Professional designers possess skills in layout, aesthetics, and user experience but commonly lack the programming background necessary to rapidly prototype fully interactive interfaces. This technical gap creates a significant bottleneck in the web development workflow. Designers spend considerable time creating visual mock-ups and prototypes, only to wait for developers to translate these designs into functional code—a process that introduces delays, miscommunications, and deviations from the original design intent [5, 6].

Existing methods for translating sketches or mock-ups into code—be they heuristic-based [7] or end-to-end approaches [8, 9]—are constrained by significant limitations. Heuristic-based systems struggle with novel or unconventional design elements, as they rely on predefined rules that cannot account for creative variations [10]. Meanwhile, end-to-end deep learning approaches suffer from limited training datasets that fail to represent the complexities of real-world design scenarios [11].

In addition, designer needs have not been adequately taken into account in traditional design-to-code systems. Designers cannot easily incorporate *style variations* (e.g., color schemes and typography) into generated code [8], as current systems typically produce generic visual styles rather than preserving the designer's aesthetic choices. Similarly, they struggle to embed *dynamic behaviors* (e.g., hyperlinks, dropdown menus) without reverting to manual development [12, 13]. This forces designers to either simplify their designs or spend additional time implementing interactive elements manually.

End users face different but equally significant barriers when attempting to create or customize web interfaces. Traditional website development requires technical knowledge that poses a substantial barrier for these individuals [14, 15]. While low-code/no-code tools have emerged as alternatives, they often present their own challenges, including limited flexibility for complex interfaces and steep learning curves [16, 17].

Current customization approaches often restrict end users to predefined options [18] or require mastery of complex configuration languages [19], frequently falling short in capturing users' genuine adaptation requirements and mental models [20]. When interfaces do not align with individual needs—which vary significantly based on context, preferences, and abilities [21, 22]—the result is suboptimal user experiences that fail to accommodate diversity.

Natural language interfaces offer a promising alternative for these users, allowing them to express their requirements without technical terminology. However, existing systems like Stylette [23] encounter difficulties with ambiguous specifications and are limited to styling adjustments through fixed sets of attributes. They typically do not support functional modifications or the creation of new interface components, significantly constraining what end users can achieve without programming knowledge.

In this context, there is a pressing need for advanced generative approaches that address the distinct needs of both designers and end users: systems that can automatically translate visual designs into functional code while preserving style and behavior for designers, and systems that enable intuitive creation and customization through natural language for end users. Addressing these challenges would significantly transform the web development landscape, making it more accessible to diverse users while enabling greater creative expression and personalization.

1.2.2 Lowering Barriers to Design Intelligent Tutoring Systems

A substantial segment of *educators* lack the programming and design expertise to create fully functional Intelligent Tutoring Systems (ITSs) interfaces on their own. Although one-on-one tutoring is known to significantly improve learning outcomes [24, 25], constructing a robust tutor interface still demands considerable technical knowledge [26, 27], limiting the potential impact of ITSs [28, 29].

Existing solutions attempt to bridge this gap through authoring tools like the Apprentice Tutor Builder (ATB) [29] and Cognitive Tutor Authoring Tools [30], which offer drag-and-drop interfaces for assembling tutor layouts. However, these tools implicitly assume educators possess interface design skills—an assumption that overlooks research showing that effective interface design requires specialized expertise [27]. While these tools mitigate the challenges of tutor model authoring, they provide limited assistance with the actual design of pedagogically effective interfaces.

This challenge represents a critical opportunity for Generative AI applications: enabling educators to transform their pedagogical expertise and teaching requirements into effective digital tutoring environments without requiring them to become interface designers or programmers. By addressing this barrier, we can significantly expand the reach and impact of personalized learning technologies.

1.2.3 Enabling Natural Interaction in Smart Home Environments

Smart home environments represent another domain where technical barriers significantly limit effective use and adoption. Traditional command-based automation systems like IFTTT[1] have required users to conform their communication to rigid system capabilities rather than allowing the system to adapt to the natural variability of human language and preferences [31–33]. This fundamental mismatch between how humans naturally express their intentions and how systems interpret commands creates significant friction in the user experience.

This challenge is particularly evident with subjective or ambiguous requests (e.g., "prepare the living room for a relaxing evening" or "make this room warmer"), which traditional systems struggle to interpret accurately in terms of actionable environmental adjustments [34–37]. While such under-specified commands are intuitively clear to humans who can leverage contextual understanding, they often lead to user frustration when systems fail to respond appropriately.

Recent advancements in smart home technology have begun exploring the integration of Large Language Models (LLMs) to enhance system responses to user

[1] https://ifttt.com/

commands. Systems like Sasha [38] and SAGE [39] utilize LLMs to improve interpretation and execution of complex or vague commands. However, these systems still primarily rely on text-based inputs to process and respond to user commands, similar to conventional smart speakers. Such textual descriptions alone may not effectively capture the full spectrum of the user's intended meaning [31, 40].

The challenge extends beyond mere comprehension of commands to establishing meaningful dialogue between users and their smart environment. When systems misinterpret ambiguous requests, users must reformulate their commands multiple times or resort to more explicit, system-friendly phrasing—adapting to the technology rather than having the technology adapt to them. This creates a communication barrier that diminishes the promise of smart environments as intuitive, responsive living spaces that enhance quality of life [41, 42].

There is a clear need for smart home interaction systems that can bridge the gap between natural human communication and precise system actions. Such systems would need to detect ambiguity in user commands, clarify intent through appropriate feedback mechanisms, and translate high-level subjective requests into specific device operations. Importantly, they should leverage multiple modalities of interaction—going beyond text to incorporate visual representations that can more effectively convey certain types of information, particularly for ambiguous or subjective concepts like "cozy," "romantic," or "energizing" room settings.

By addressing these challenges, we can transform smart home interfaces from rigid command interpreters to intuitive communication partners that adapt to users' natural expression patterns. This would significantly lower the adoption barrier for smart home technologies while increasing their utility and satisfaction across diverse user groups, ultimately fulfilling the promise of truly responsive living environments that enhance comfort and quality of life.

1.2.4 Research Opportunities and Thesis Focus

The challenges outlined in the previous sections reveal significant opportunities for innovation at the intersection of AI and interactive systems design. This dissertation explores these opportunities across three domains where AI can democratize creation and customization capabilities.

For *Web Development*, we pursue two complementary opportunities with distinct user-centered focuses: first, to develop visual-to-code translation systems that emphasize designers' specific needs for creative control, style preservation, and dynamic behavior specification that current systems neglect; second, to create natural language interfaces that prioritize end-users' mental models and customization desires beyond the limited options currently available. Rather than forcing designers to compromise their vision or requiring end-users to learn technical skills, these opportunities represent a fundamental shift in how we approach web development—placing human creativity and expression at the center while technology adapts to support it.

For *Intelligent Tutoring Systems*, we identify the opportunity to create AI-assisted authoring tools that leverage educators' pedagogical expertise while automating interface design decisions. This approach offers multiple potential benefits: improving the usability of authoring tools could significantly increase adoption rates among educators who have previously avoided digital tutoring systems due to technical barriers; wider adoption would extend the reach of personalized tutoring to more diverse student populations; and better-designed interfaces resulting from AI assistance could enhance student engagement, knowledge retention, and overall learning outcomes. The opportunity lies in transforming intelligent tutoring from a specialized technical domain to an accessible teaching tool that any educator can implement effectively.

For *Smart Home Environments*, we explore the opportunity to leverage recent advances in large language models' representational power and multimodal understanding capabilities for everyday interaction contexts. As LLMs have demonstrated increasingly sophisticated comprehension of nuanced human language and the ability to connect concepts across modalities, a significant opportunity exists to apply these capabilities to the gap between natural human expression and precise device control. This opportunity extends beyond mere command interpretation to rethinking how humans might communicate with their environments—moving from explicit command structures to more intuitive, contextual, and multimodal conversations that feel natural to users.

These research opportunities guide the contributions presented in subsequent chapters, each demonstrating how AI can serve as a bridge between domain expertise and technical implementation. By focusing on maintaining user agency throughout the creation and customization processes, our work aims to expand who can partici-

pate in shaping interactive systems while respecting their domain knowledge and creative intentions.

1.3 Main Thesis Contributions

Building on the research opportunities identified above, this dissertation presents novel methods and tools that employ AI to address the gaps between domain expertise and technical implementation. Our contributions span three primary application domains—web development, intelligent tutoring systems, and smart home environments—addressing different user expertise levels and dimensions of interaction. These contributions, categorized in Table 1.1 according to their technical category, target user expertise level, and interaction dimension, demonstrate concrete implementations of our vision across diverse domains.

As shown in Table 1.1, our contributions address distinct application domains while targeting both professional users with domain expertise and end users without technical backgrounds. These contributions span the full spectrum from creation of new interactive systems to customization of existing ones, providing solutions tailored to each user category's unique challenges and capabilities. The following sections detail each contribution across web applications, intelligent tutoring systems, and smart home environments.

1.3.1 Algorithmic Approaches for Professional Designers

In the domain of web development, our contributions address both professional designers and novice end-users, focusing on different aspects of the creation and customization process.

Transformer-Based Design-to-Code

Professional designers often create visual mockups but must wait for developers to implement these designs as functional code, leading to costly iterations and potential miscommunications. We developed a transformer-based architecture that automatically translates visual designs into code, allowing designers to rapidly prototype and test their interfaces.

Table 1.1 Thesis Contributions Overview

User Category	Contribution	Application Domain
Professional	**Design-to-Code model and dataset advancement:** A novel architecture that translates visual mockups into functional code with higher accuracy and generalization (Chapter 2)	Web Development
	Sketches and Interactive Behavior to Web UIs: Techniques for style-aware sketch-to-code conversion and incorporating dynamic behaviors into design sketches through intuitive annotations (Chapter 2)	Web Development
	Pedagogical Goals to Tutors' UIs: A system that automatically converts educators' teaching requirements into interactive tutoring interfaces (Chapter 3)	Intelligent Tutoring Systems
End-User	**Text to Web UIs:** A natural language interface that enables non-technical users to create websites through conversation (Chapter 4)	Web Development
	Integrating GenAI into Dynamic UIs: A framework allowing users to modify existing interfaces through structured natural language commands (Chapter 4)	Web Development
	Multimodal Disambiguation in Smart Homes: A method for clarifying ambiguous smart home commands with multiple visual and textual interpretations (Chapter 5)	Smart Home Environments

Unlike previous approaches that relied on LSTM-based models, our transformer architecture leverages attention mechanisms to better capture relationships between visual components and their corresponding code representations. To overcome the

limitations of existing datasets, we created a specialized scraping pipeline to curate and clean online code, building the WebUI2Code dataset with 8,873 screenshot-code pairs. We also generated a synthetic dataset of Bootstrap websites to further enhance model training.

Our architectural evaluation demonstrated that the transformer-based approach outperforms traditional LSTM-based models when tested on web UI datasets, showing better handling of complex layouts and improved generalization to diverse design styles. This contribution significantly accelerates the design-to-implementation workflow, enabling designers to iterate more rapidly on their concepts.

Style-Aware Sketch-to-Code

Building on the design-to-code translation, we addressed a specific limitation of existing approaches: their inability to capture and implement style variations. Current systems typically generate code with generic visual styles rather than preserving the designer's aesthetic choices.

We developed an algorithm that allows designers to not only translate sketches into code but also specify the visual style using reference images. The system segments the input sketch to identify components and their positions, then applies styles from the reference image using two techniques: (1) a clustering-based approach to extract prominent colors, and (2) a feature distance-based metric to select appropriate text styles.

Our experiments with navigation bar components demonstrated that this approach effectively captures and applies style references, enabling designers to explore different aesthetic directions directly from their sketches. This contribution enhances the creative control designers maintain throughout the automation process.

Dynamic Behavior in Sketch-to-Code

Complementing the style-aware approach, we also addressed the challenge of specifying interactive behaviors in early design sketches. Existing systems could translate sketches into static layouts but required manual implementation of dynamic elements like dropdown menus, links between pages, and interactive components.

We introduced a set of symbolic annotations that designers can incorporate directly into their sketches to represent dynamic behaviors. Some symbols leverage established visual languages (e.g., downward arrows for dropdown menus), while others were introduced specifically for this purpose. A convolutional neural network identifies these symbols and their positions, enabling the generation of code that includes the specified interactive behaviors.

This contribution eliminates the need for separate implementation of dynamic elements, allowing designers to specify both structure and behavior in a single sketch. By preserving the designer's intended interactions from the earliest design stages, this approach further streamlines the prototyping process.

1.3.2 Empowering Educators through AI-Assisted Tools for Instructional Design

Generative AI for Intelligent Tutoring Systems

Our first contribution addresses the challenge educators face when creating intelligent tutoring systems. While educators possess domain knowledge and pedagogical expertise, they often lack the technical skills needed to design effective digital interfaces. We developed a tool that uses Generative AI to bridge this gap, allowing educators to transform their teaching requirements into functional tutoring interfaces.

Unlike previous approaches that either required technical expertise or provided limited design assistance, our system introduces two key innovations: **AI-assisted pedagogical step decomposition**: The system automatically breaks down high-level tutoring goals into structured instructional actions (e.g., concept introduction, guided practice, feedback loops) derived from established ITS strategies. This decomposition serves as the foundation for interface generation and ensures alignment with effective pedagogical practices. Educators can modify these steps to maintain control over the tutoring process.

Preference-driven UI refinement: Rather than generating a single interface design, our system produces multiple diverse interface drafts. Educators can then express their preferences through direct interaction with these drafts, selecting elements they prefer. The system uses these preferences to generate a final interface that aligns with the educator's vision while benefiting from AI-driven design automation.

Through a user study with eight K-12 educators, we demonstrated that our approach significantly enhances user satisfaction and interface quality compared to traditional drag-and-drop methods, without extending design time. This contribution maintains the educator's agency in the design process while lowering the technical barriers to creating effective tutoring interfaces.

1.3.3 Web Development for End Users: Natural Language Tools for Creating and Customizing Websites

LLM-Driven End-User Web Development

For novice users without design or programming expertise, we developed a tool that leverages Large Language Models to enable website creation through natural language. Traditional website development requires technical skills that present a significant barrier for most users, while existing low-code/no-code tools often have steep learning curves and limited flexibility.

Our approach centers around prompt engineering, which constrains the LLM to follow a predefined template structure. This enables users to express their requirements in natural language and iteratively refine the generated code without understanding the underlying implementation. Key innovations include:

1. **Structured template for LLM output**: By constraining the model's responses to follow a specific structure, our system can reliably parse and modify generated code, maintaining consistency across iterations.

2. **Context-preserving refinement strategy**: Our system implements a modification strategy that updates only specific parts of the code rather than regenerating the entire document, preserving context and reducing token usage.

3. **Multi-page support**: Unlike previous approaches, our system enables users to create and link multiple pages, supporting more complex website structures.

A proof-of-concept implementation demonstrated that this approach significantly reduces the technical knowledge required for website creation, making web development accessible to a broader audience. By requiring minimal technical expertise,

our approach eliminates the need to master programming languages or design tools, democratizing the creation process.

Real-Time Web Customization

Traditional graphical user interfaces often offer limited customization options through predefined settings, constraining users' ability to adapt interfaces to their specific needs. We developed MorphGUI, a framework that combines traditional GUI controls with LLM-powered customization to enable real-time interface modification through natural language.

MorphGUI introduces a structured input approach that separates stylistic ("how it should appear") from functional ("what it should do") modifications, guiding users through the customization process. This structured approach helps address ambiguity issues commonly found in open-ended natural language interactions while maintaining flexibility. The system can dynamically generate new interface elements or modify existing ones, providing customization capabilities beyond predefined settings.

Through an experiment with 18 participants, we demonstrated that users regardless of their technical expertise could effectively personalize interfaces by expressing desired changes through natural language inputs. This contribution empowers users to realize customization intentions that may not have been anticipated by designers, enhancing the adaptability of interactive systems.

1.3.4 Smart Home Disambiguation for Everyday Users

In smart home environments, users often issue subjective or ambiguous commands (e.g., "make the room cozier") that traditional systems struggle to interpret accurately. We developed a multimodal disambiguation approach that combines visual and textual cues with natural language commands to improve system understanding and user satisfaction.

When a user issues a command that the system detects as ambiguous, our approach generates multiple possible interpretations and presents them to the user through both textual descriptions and visual representations. The user can then select the option that best matches their intent, allowing the system to take appropriate ac-

tion. This interaction model prevents misinterpretations and frustration by involving the user in resolving ambiguity.

Through a user study with seven participants, we evaluated the effectiveness of textual versus visual disambiguation approaches, finding that both improved accuracy in system interpretations. This contribution enhances the naturalness of interaction in smart environments while maintaining precise control over system actions.

1.3.5 Summary

As illustrated in Table 1.1, the main thesis contributions address different application domains and user categories through a combination of algorithmic approaches and complete tools. This positioning of contributions—spanning from professional designers creating web interfaces to everyday users controlling smart homes—allows us to address technical barriers across the full spectrum of interactive system development. The subsequent chapters will explore each contribution in depth, examining both implementation details and their impact on lowering barriers to technology creation.

1.4 Document Structure

The remainder of this dissertation is structured as follows:

- **Chapter 2: Algorithmic Design-to-Code Translation for Professional Designers** explores our transformer-based architecture for converting visual designs to code with style preservation and dynamic behavior specification.
- **Chapter 3: Empowering Educators: AI-Assisted Tools for Instructional Design** presents our work on intelligent tutoring systems for transforming pedagogical requirements into effective digital interfaces.
- **Chapter 4: Web Development for End Users: Natural Language Tools for Creating and Customizing Websites** details our approaches to LLM-driven website creation and real-time interface customization through natural language.

- **Chapter 5: Intuitive Home Control: Disambiguation Methods for Everyday Users** addresses resolving ambiguity in natural language commands through visual and textual cues.
- **Chapter 6: Discussion and Future Work** synthesizes insights across domains, discusses limitations, and outlines future research directions.
- **Chapter 7: Conclusion** summarizes the key contributions and broader impact of this dissertation.

This structure organizes contributions according to both user categories and approach types, highlighting how each chapter addresses specific barriers faced by different user groups while maintaining focus on the goal of democratizing interactive system creation and refinement.

Chapter 2

Algorithmic Design-to-Code Translation for Professional Designers

Publication notice. Parts of this chapter expand upon and unify material previously published in the following peer-reviewed venues: *EICS'25* ("Advancing Code Generation from Visual Designs through Transformer-Based Architectures and Specialized Datasets"), *EICS'22 Companion* ("Style-Aware Sketch-to-Code Conversion for the Web"), and *PAINT'22* ("Creating Dynamic Prototypes from Web Page Sketches").

2.1 Introduction

Web applications have become essential for establishing and maintaining an effective online presence for both individuals and organizations. The virtual representation of an entity, whether a personal blog or corporate website, plays a critical role in creating first impressions and ensuring sustained audience engagement. Due to their significance, there is growing demand for sophisticated, appealing, and modern websites, leading to increasingly complex design and development processes.

The development of web interfaces typically progresses through an iterative process involving multiple artifacts: hand-drawn sketches, wireframes, and mock-ups [43–45]. Sketches serve as initial, often rudimentary representations that allow designers to swiftly visualize ideas. As noted by designers, "sketches allow designers to focus on basic structural issues instead of unimportant details" [46], making them

particularly valuable during early exploration. Wireframes identify the positioning of UI elements without styling or colors, resembling blueprints. Mock-ups present more refined versions with styling, graphics, colors, typography, and other intricate visual details. Once these designs receive approval from stakeholders, web developers proceed with implementation.

This traditional workflow creates significant challenges, as documented in extensive studies of design and development practices [5, 6]. Since design and implementation typically involve separate teams, the process from concept to completion becomes not only time-consuming but also expensive. The back-and-forth between designers and developers introduces delays, potential miscommunications, and deviations from the original design intent. User studies confirm that while rough prototypes can identify the same usability problems as finished ones [47, 48], the manual transition from sketch to code remains a major bottleneck [12, 13].

To address these challenges, researchers have explored various approaches to automatic website generation [49, 7]. These approaches aim to automatically convert design artifacts into functional code, reducing the need for manual coding and streamlining the transition from design to implementation. This integration of design and implementation phases ensures a more direct translation of design intentions into code, resulting in faster delivery times, cost savings, and fewer opportunities for miscommunication. Furthermore, automated website creation democratizes web development, allowing individuals without technical expertise to create professional-grade websites, fostering a more inclusive digital environment where creativity is not bounded by technical constraints.

Two primary modalities for automatic website generation have emerged: mock-up-driven approaches that convert detailed mock-ups into code, and sketch-driven approaches that transform preliminary hand-drawn sketches into functional websites. The sketch-driven approach is particularly beneficial for beginners, offering those unfamiliar with web development processes an opportunity to transform basic sketches into operational websites. Sketches serve as a natural means of human-AI interaction, harnessing the inherent human ability to visualize and express ideas through simple drawings regardless of technical expertise [6]. For designers, sketch-driven conversion allows rapid testing of interactive prototypes, facilitating faster iteration and refinement of concepts.

These approaches can be further categorized into heuristic-based methods [7] that rely on predefined rules and patterns, and end-to-end approaches [11] that use deep learning models to handle the entire conversion process. Heuristic-based systems process sketches and mock-ups by leveraging predefined sets of rules and patterns, making decisions based on established guidelines. For example, a rectangular shape might be recognized as a button, and parallel lines might be interpreted as text fields. However, these systems can lack flexibility, and their effectiveness is heavily tied to the quality of their underlying rules. If a design contains novel or unique elements, the heuristic model might misinterpret or fail to detect them entirely.

In contrast, end-to-end approaches utilize deep learning models to manage the entire conversion process. Instead of operating with fixed rules, these models are trained on vast datasets comprising various designs and their corresponding outputs. With sufficient training data, they can continuously adapt and improve [50], keeping pace with evolving design trends and identifying subtle patterns that might not be explicitly defined in heuristic rules. The encoder-decoder framework common in these methodologies typically employs convolutional neural networks (CNNs) to analyze image features and recurrent neural networks (RNNs) to generate corresponding code. Pix2code [8] pioneered this approach, translating screenshots into domain-specific language representations that could be compiled into HTML code.

Despite progress in this field, significant limitations remain in existing design-to-code systems. First, their performance is hindered by the limited availability of specialized training datasets. Many existing UI/code datasets lack diversity, being overly simplistic and inadequately representative of real-world complexity. Second, current approaches fail to consider aesthetic aspects (colors, typography, shadows) of the generated interface, producing only basic structure without preserving designers' stylistic choices [8]. As noted by researchers, "these works capture well the backbone structure of the GUI and translate it into code, but they are not designed to consider the aesthetics (e.g., colors and shadows) to be applied to the generated interface" [6]. Third, dynamic behaviors (links, dropdown menus, page transitions) typically require separate manual implementation, preventing designers from specifying interactive elements in their initial sketches. Finally, current end-to-end approaches predominantly rely on LSTM-based architectures, with the potential of transformer-based models largely unexplored in this context.

This chapter presents our research addressing these limitations through three interconnected innovations. First, we introduce a transformer-based architecture for design-to-code translation that leverages attention mechanisms to better capture relationships between visual components and their code representations. By developing specialized web scraping pipelines and synthetic datasets, we overcome the data limitations that have constrained previous approaches. Second, we present a style-aware approach that allows designers to specify visual aesthetics through reference images, automatically extracting and applying style elements including colors and typography. Finally, we introduce a method for specifying dynamic behaviors directly in sketches through symbolic annotations, enabling the generation of fully interactive prototypes without separate implementation steps.

Our work advances the state of the art in design-to-code translation by addressing both the technical and user-centered aspects of the problem. By improving the architectural foundation through transformer-based models, we enhance the accuracy and generalization capabilities of automated translation. By incorporating style awareness and dynamic behavior specification, we preserve more of the designer's creative intent throughout the automation process. Together, these contributions significantly reduce the technical barriers that have traditionally separated design from implementation, empowering designers to rapidly prototype and iterate on their concepts without waiting for manual code translation.

The following sections detail our transformer-based architecture for design-to-code translation, our style-aware sketch-to-code conversion approach, and our method for specifying dynamic behaviors in sketches. Through empirical evaluations and case studies, we demonstrate the effectiveness of these approaches in streamlining the web development workflow while preserving designer control over both stylistic and functional aspects of the resulting interfaces.

2.2 Transformer-Based Architecture for Design-to-Code Translation

We introduce three interconnected components aimed at advancing the automation of web development from visual designs and overcoming the constraints of existing datasets and methodologies. Initially, a web scraping pipeline is established to

gather and process actual website code and screenshots. This pipeline acquires the HTML content rendered by websites and goes through a sanitization process to eliminate unwanted code elements such as scripts and comments. It also identifies and downloads any related CSS files, which are then processed by a custom parser. The processed HTML and consolidated CSS files are subsequently rendered to generate website screenshots. A classifier is utilized to filter these screenshots, retaining only those that maintain the original layout post code reduction.

Subsequently, a synthetic dataset generation technique is implemented. This method generates websites in a procedural manner while adhering to a popular front-end framework, providing control over parameters such as layout, components, color palettes, and text. A recursive screenshot capture system is integrated to ensure comprehensive rendering of all page elements. To produce a sketch variant, UI components are replaced with hand-drawn sketches based on bounding box annotations.

Finally, we compare the performance of two transformer models for the design-to-code task: a Pix2Struct model and a larger, more powerful Gemma2b model. Both models utilize a vision transformer encoder to process rendered website screenshots, coupled with an autoregressive transformer decoder that produces HTML and CSS tokens sequentially. The evaluation of these models is conducted on a real-world dataset from a web scraping pipeline and on synthetic datasets under various conditions. These conditions encompass scenarios where rendered elements are substituted with sketches to mimic lower-fidelity designs, different levels of website layout complexity, and variations in token thresholds for code generation. This assessment enables us to examine the performance discrepancies between the Pix2Struct and Gemma2b models in diverse design-to-code scenarios.

2.2.1 WebUI2Code Dataset: Real-World Website Screenshot-Code

Our data collection procedure utilizes a multi-stage pipeline to fetch, sanitize, and validate websites, ensuring high structural fidelity and practical manageability for subsequent machine learning tasks. As shown in Figure 2.1, the process starts by loading each target URL in headless mode using Selenium and Google Chrome. The browser operates at a fixed resolution (1280×1024) for standardized captures.

A retry mechanism is activated if a site fails to load completely. Common pop-up dialogs (e.g., cookie consent banners) are automatically dismissed by removing their scripts or simulating clicks on typical acceptance buttons.

After obtaining the raw HTML, an automated sanitizing phase follows, where extraneous or dynamic tags—such as `script`, `meta`, `noscript`, `svg`, and `iframe`—are systematically eliminated. The sanitization script also corrects malformed tags and replaces asynchronous attributes (e.g., `data-src`, `data-lazy-src`) with their standard counterparts (`src`, `srcset`). Additionally, all external images are replaced with a standard placeholder image to avoid unnecessary resource downloads and maintain visual consistency. To reduce minor layout differences, certain HTML elements (e.g., `<ol>` tags) are converted to functionally similar ones (e.g., `<ul>`) when essential structural differences are not present. Subsequent to this filtering, a cleaning tool (`clean-html`) eliminates any remaining comments, irregular whitespace, or unnecessary line breaks, and formats the code with a consistent indentation scheme.

Upon obtaining the sanitized HTML, the process identifies external CSS references, downloads each `.css` file, and parses them using a custom extension built on top of `tinycss2`. This parser recognizes at-rules and qualified rules, enabling selective elimination of styles that do not correspond to any tags or classes present in the cleaned HTML. Properties deemed purely ornamental or irrelevant to the core structure of the website (e.g., transition animations or outdated browser-specific rules) are pruned to reduce complexity. The remaining valid CSS is then consolidated into a single file, and the HTML references are adjusted accordingly.

Prior to the final screenshot capture, a *web framework detector* is applied to identify frameworks like React, Gatsby, Nuxt, Backbone, or Next. These frameworks typically add substantial client-side code that may fail to render correctly post-sanitization. If any of these frameworks are detected, the respective site is excluded from the final dataset to ensure consistency. Additionally, sites with sanitized HTML containing no CSS classes or being trivially empty are also filtered out.

After applying these filters, screenshots are generated by loading the local HTML file in a headless browser environment. By rendering the sanitized HTML along with the consolidated CSS, the resulting screenshot more accurately represents the final "minimal" version of the webpage compared to capturing the original online page. Examples of the resulting mockups are illustrated in Figure 2.2. The reduction in

size is significant, especially in the CSS: Table 2.2 demonstrates that many pages decrease from tens of thousands of lines to just a few thousand or fewer.

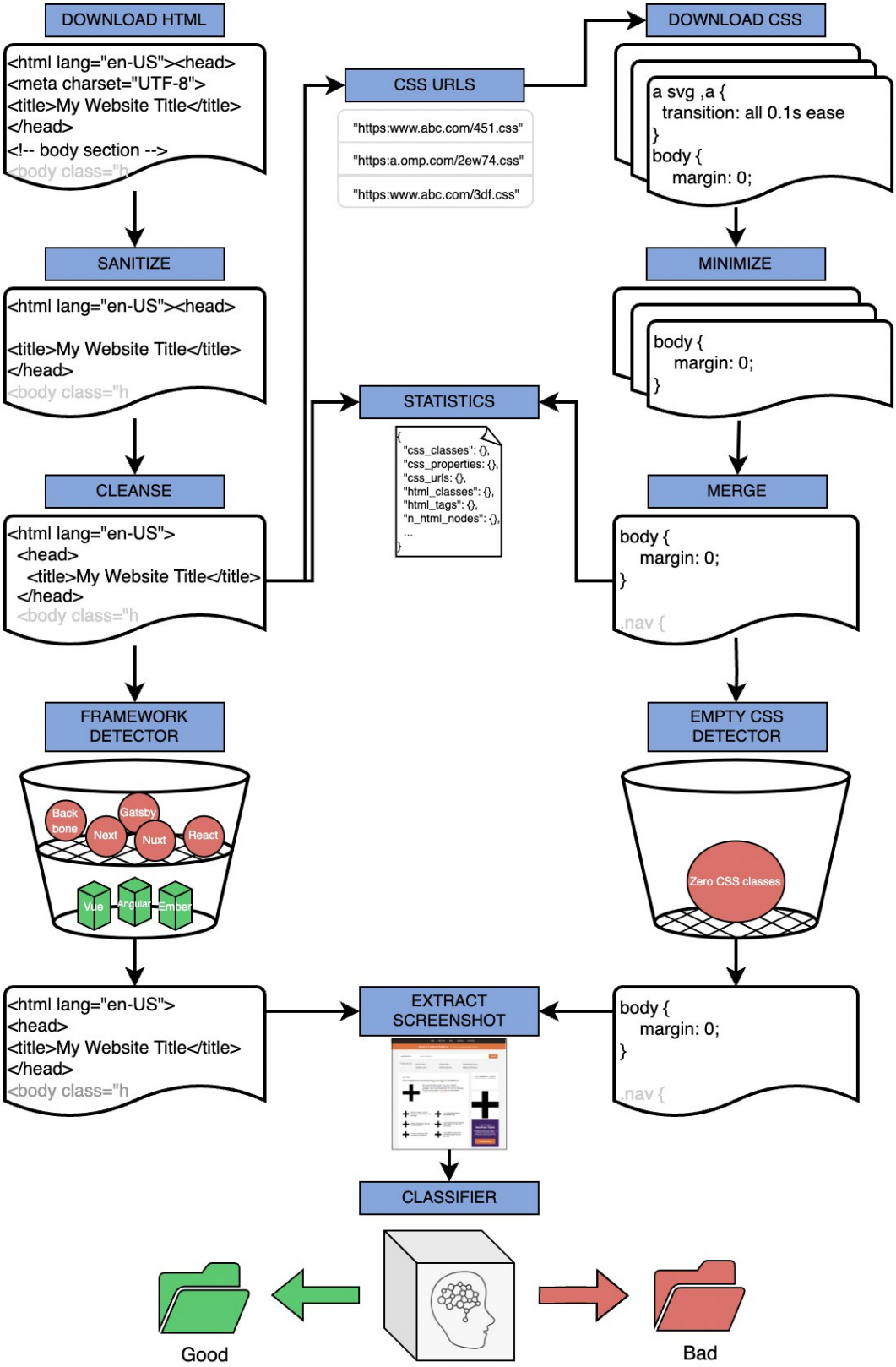


Fig. 2.1 The diagram depicts the step-by-step process of acquiring website code and screenshots. Initially, the HTML file is downloaded, sanitized, and cleansed. Next, the HTML file is analyzed by a web framework detector that screens files with specific frameworks. Subsequently, CSS URLs are identified from the HTML file, and the corresponding CSS files are downloaded, minimized, and merged. A detector is then employed to filter out files with no CSS classes. The processed HTML and CSS files are then utilized to generate website screenshots. Finally, a classifier assigns labels to the screenshots based on their quality.

A final quality assessment is conducted using a ResNet50 convolutional neural network [51] that has been fine-tuned on a labeled set of "good" vs. "bad" screenshots. This binary classifier was trained to effectively differentiate between pages that maintain essential layout integrity and those that display significant structural issues or missing styles. The training data for this classifier was obtained from initial pilot studies, where grades (ranging from 0 to 5) were manually assigned based on visual quality and structural completeness. Lower scores were often associated with empty pages, heavily distorted layouts, or frameworks that were beyond the capabilities of the pipeline. By translating these manual ratings into "good" and "bad" categories, the classifier achieved an accuracy of over 80% on a separate test set. Following this classification step, the "bad" images are removed, resulting in a refined collection of "good" screenshots along with their processed HTML/CSS for further analysis.

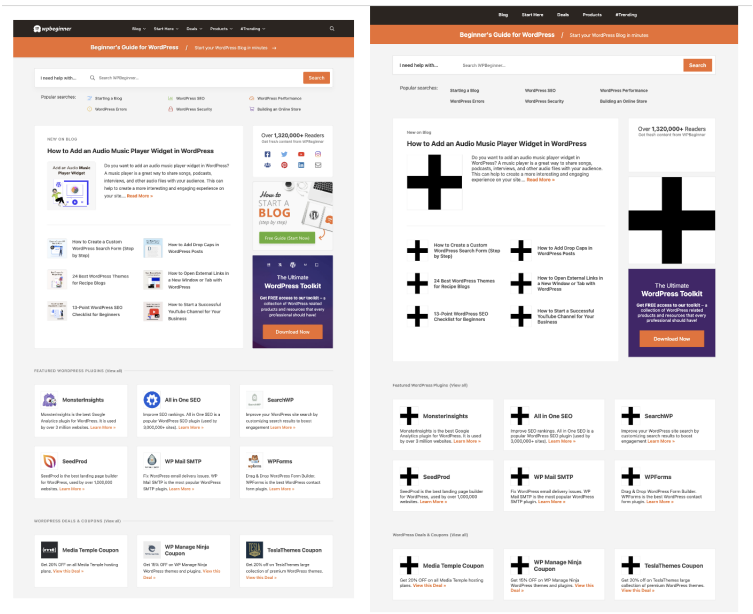


Fig. 2.2 Screenshot of website before and after processing.

Three initial investigations examined this process on subsets of different complexities: an initial trial on a small list of 51 blog sites, a second on 100 sites from the Majestic Million ranking, and a third on 100 `.blog` domains. These step-by-step tests not only emphasized the strengths of the process (particularly for simpler or blog-focused pages) but also identified challenging scenarios (e.g., large commercial sites with anti-scraping measures or sophisticated client-side rendering).

	Total
Errors	29736
Excluded	16459
Bad images	19716
Good images	34089
TOT	100000

Table 2.1 Results obtained by filtering websites according to their quality

Human evaluators compared the processed screenshots to their original online counterparts, assigning numeric grades and analyzing major differences. The results informed thresholds for excluding particular frameworks or "unstructured" pages. When extended to 100 000 domains from the Majestic Million, the pipeline retained approximately 34,000 "good" pages out of 100,000 total entries; see Table 2.1 for an overview. Overall, the biggest sources of error were connection and SSL issues, especially from highly-protected websites, along with malformed CSS.

	Raw	**Processed**
CSS Classes	1788.03	143.39
CSS Classes Skipped	0	1596.25
CSS Properties	220.31	78.42
CSS Properties Skipped	0	17.83
CSS URLs	8.14	8.11
HTML Classes	238.64	234.75
HTML Tags	35.94	26.86
HTML Nodes	860.75	699.58
Lines CSS	20240.5	2255
Lines HTML	1542.25	996.08

Table 2.2 Summary of average statistics collected for each website

Through a systematic refinement of the pipeline at each stage—from HTML retrieval and sanitization, to CSS minimization and screenshot classification—the WebUI2Code dataset is able to produce a relatively clean and structurally representative set of real-world UIs. This multi-tiered filtering strategy intentionally sacrifices coverage in order to filter out visually unrecognizable or fundamentally broken pages. While this filtering improves quality, it also biases the dataset towards simpler, more statically structured websites (such as blogs and informational pages) and tends to exclude highly dynamic, JavaScript-heavy web applications, which

represents a known limitation in coverage. The end result is a large-scale collection of HTML/CSS/screenshot triples that exhibit significantly less noise compared to raw scrapes, providing a valuable test environment for design-to-code models and other interface-centric research.

2.2.2 HTML Bootstrap Synthetic Dataset

The HTML Bootstrap Synthetic Dataset was created utilizing the open-source tool WebGenerator [52]. This tool was designed for generating synthetic web-based user interfaces (UIs) using the Bootstrap framework. It offers various options for generation, such as probabilities for layouts and sections, sizes of screenshots, and types of components. These components encompass commonly used elements like Cards, Placeholders, Tables, and Navigation bars, as well as specialized ones like Carousels and Forms. The tool populates these elements with random "lorem ipsum" sentences to simulate the appearance of practical UIs. Additionally, it can produce diverse website styles by choosing random color palettes and adjusting the CSS file in the HTML code, enabling the creation of multiple unique website designs. By adjusting the parameters of the WebGenerator, the resulting websites exhibit a wide range of designs and structures. This variability ensures that any model trained on this dataset will possess increased robustness and generalizability. Furthermore, due to its utilization of the Bootstrap framework, this dataset remains relevant in contemporary web development. Bootstrap stands as one of the most popular front-end libraries, with its components being widely used on numerous websites. Therefore, it holds practical significance and can serve as a valuable resource for research aiming to develop real-world applications. To tailor the tool to our requirements, we implemented a recursive system for capturing screenshots to ensure that all webpage elements are captured within the screenshot.

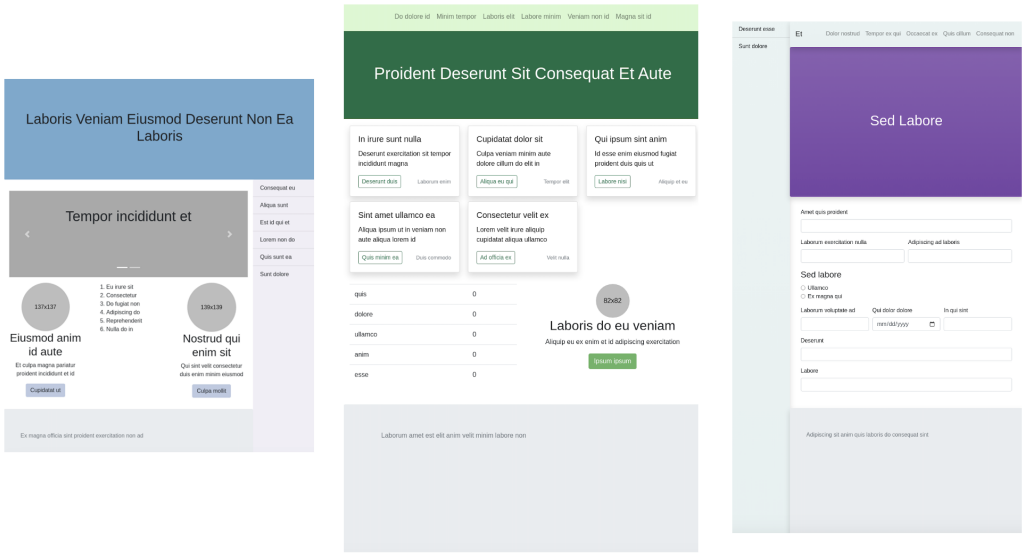


Fig. 2.3 A series of random sampled web interface mock-ups from synthetic dataset demonstrating varied design patterns. From left to right: a content-focused layout with distinct navigation elements; a grid-based interface highlighting data presentation; and a minimalistic approach with prominent call-to-action features.

Examples of the website screenshots generated by the WebGenerator are shown in Figure 2.3. As these screenshots are artificially created, they do not include any real-world data, making them a valuable dataset for research purposes without privacy or data rights concerns. The JSON annotations provided with details on GUI regions and types offer researchers a comprehensive insight into the webpage's layout and design elements. Such annotations can be beneficial for supervised machine-learning tasks that require accurate labels. We revised the component annotation process to identify more specific components like text and buttons instead of general labels such as 'header' or 'footer. This adjustment facilitated the conversion to sketches. After configuration, a dataset of 50,000 samples was generated. Each sample comprises a PNG image of the webpage screenshot, its corresponding HTML code, and JSON annotations.

Description	Value
Probability that the Sidebar is present	0.7
Probability that the Header is present	0.5
Probability that the Navbar is present	0.8
Probability that the Footer is present	0.6
Probabilities of each layout (4 layouts).	0.25
Probability that the page's Body is boxed inside a container	0.0
Probability of having a big header	0.0
Probability of the Sidebar being at the left side of the Body	0.8
Probability of the Navbar being above the header	1.0

Table 2.3 Generation probabilities used for the synthetic dataset configuration

2.2.3 Generate Synthetic HTML Bootstrap Dataset Sketches

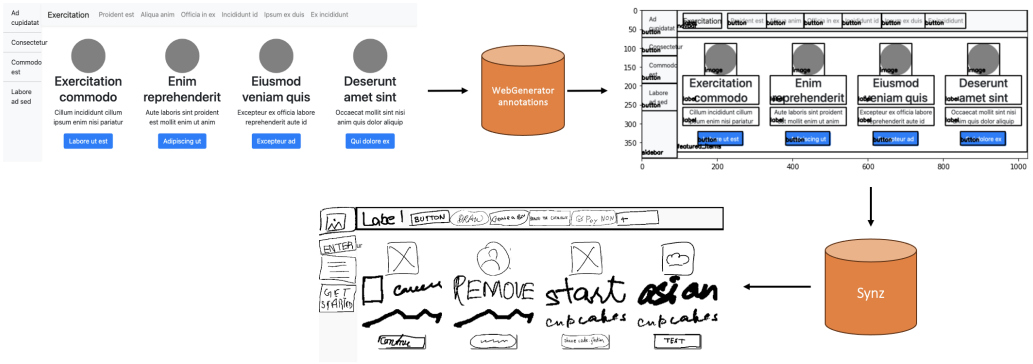


Fig. 2.4 This flow diagram illustrates the multi-stage transformation process, which commences with synthetic webpages containing structured design elements and textual content. Subsequent extraction of specific components results in generator annotations. These annotations facilitate the conversion of structured designs into abstract, hand-drawn sketches.

In order to apply the synthetic dataset in the sketch-to-code challenge, we enhanced the generation function by incorporating a technique to track the bounding boxes of detailed components, the fundamental design elements. Due to the synthetic nature of the dataset, we were able to annotate and retrieve these bounding boxes easily without the need for object detection methods. Subsequently, the identified components were replaced with their sketched counterparts from the Synz dataset [53], a comprehensive compilation of sketched web components. Each component is cropped to the smallest rectangle containing the first non-white pixel, as we noticed redundant whitespace in the Synz dataset. Moreover, one component out of the

top 10 components with a similar aspect ratio to the bounding boxes is randomly chosen for inclusion. This methodology facilitated the development of a synthetic dataset of web sketches with varied compositions. As a result, this iteration generated almost 10,000 websites (specifically 9,789), following the same methodology and parameters as the original synthetic Bootstrap dataset.

2.2.4 Transformer Architecture for Design-to-Code

Transformer models have revolutionized sequence-to-sequence translation tasks by utilizing self-attention to capture long-range dependencies without the sequential processing limitations of LSTM-based methods [54]. In the context of design-to-code translation, a transformer is capable of processing image patches (visual tokens) and partial code tokens (textual context), which results in the generation of new tokens representing the evolving HTML or DSL output (Figure 2.5). Here, we provide a brief overview of our chosen approach, highlighting practical considerations that are important when dealing with large screenshots and extensive code sequences.

Vision Transformer Encoding. In line with the methodology of Pix2Struct [55], the input screenshot undergoes an initial division into patches of equal size (e.g., $P \times P$ pixels), each of which is transformed into a latent vector. By stacking transformer layers over these patch embeddings, a contextualized representation of the webpage image is obtained, capturing both the overall layout and specific component details. Unlike architectures based solely on convolutions, this patch-wise strategy inherently treats the layout as a sequence of visual tokens, allowing for adaptable handling of various aspect ratios and user interface complexities.

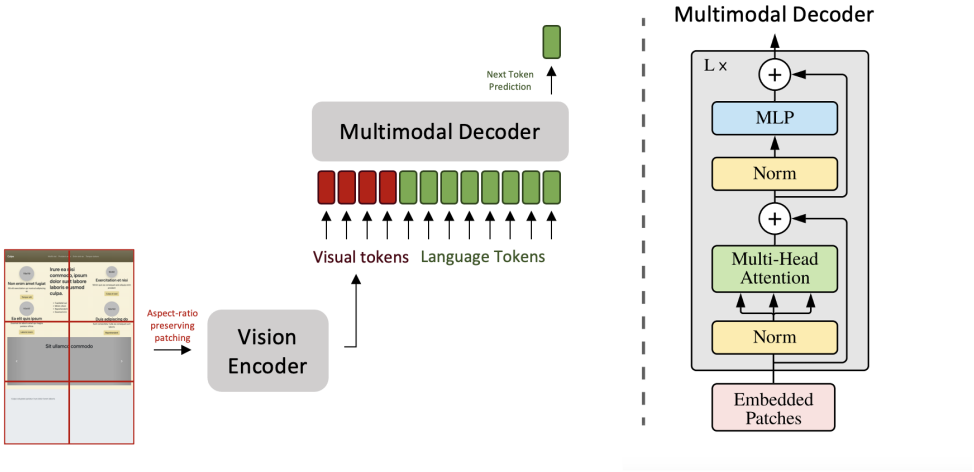


Fig. 2.5 A visualization of the process to transform web mock-ups into code. On the left, a web mock-up is divided into patches, which are handled by Vision Transformers to produce contextualized encodings. The middle section showcases the autoregressive transformer decoder, which creates code tokens in a sequence according to the visual encodings.

Utilizing Sliding Window for Generating Lengthy Code. When creating the complete HTML content of a webpage, it often surpasses the standard context-length limits for Pix2Struct [55]. To address this challenge, we implement a *sliding window* strategy for text generation: at each iteration, the decoder is provided with (i) the full set of image embeddings, and (ii) a truncated history of previously generated tokens (e.g., the last N tokens). It then predicts the next M tokens, shifts the window, and repeats the process. Through empirical observations, we have found that this approach maintains coherence as the model learns to seamlessly continue from its previous state, associating specific image regions with partial code snippets. Although each window does not encompass the entire code history, the combination of local textual context and consistent image encoding is adequate for maintaining syntactic coherence. This technique effectively overcomes GPU memory limitations while still allowing for the reconstruction of lengthy HTML files.

In contrast, FerretUI-Gemma has the capability to handle larger textual contexts in a single forward pass, thereby relying less on sliding windows and more on a comprehensive memory of the entire code sequence. While this approach often leads to improved performance on very large or heavily annotated web pages, it demands sig-

nificantly more computational resources. On the other hand, our approach based on Pix2Struct, combined with sliding-window text decoding, strikes a balance, enabling the processing of intricate layouts without exceeding memory constraints. Both methodologies emphasize the significance of accurately modeling *image-to-code ratio awareness*: the network must align different sections of the UI screenshot with semantically relevant code segments, whether this alignment is achieved iteratively (sliding window) or in a single comprehensive pass (long context). As confirmed by our experiments, both strategies can be effective, but practitioners must consider the trade-off between hardware resources and the complexity of the web UIs being analyzed.

2.2.5 Experiments

In the following sections, we assess the performance of the transformer-based Pix2Struct architecture in three different scenarios: validating our real-world WebUI2Code dataset and synthetic datasets using the model's performance, and comparing Pix2Struct's effectiveness for design-to-code generation with traditional RNN-based methods on established benchmarks. These experiments offer insights into both the capabilities and constraints of transformer-based approaches for automated web development, with implications for practical web design applications.

Datasets

The experiments utilized the following datasets to evaluate the proposed method:

WebUI2Code Dataset: The "WebUI2Code" dataset, described in Section 2.2.1, arises from our scraping pipeline designed to extract website codes and their corresponding screenshots from the web. Comprising 8,873 samples, each data point in this dataset consists of an HTML file, an associated CSS file, and a screenshot. Alongside these, the dataset also includes supplementary files, such as a JSON detailing the metrics from the scraping process, and the unprocessed versions of the HTML and CSS files. The WebUI2Code dataset was segmented into distinct versions to adapt to different experimental needs, grounded in the dataset's size and complexity. By creating subsets of the data with specific token thresholds, we aim to ensure that the experiments are both manageable and scalable.

The WebUI2Code-4096 Version has been set with a token threshold of 4,096. This version includes 2,442 samples and has been curated to align with the token constraints applied to other datasets in parallel studies. Next, the WebUI2Code-8192 Version increases the token threshold to 8,192, accommodating slightly more intricate samples. Composed of 1,906 samples, the number of usable samples in this version varies based on additional constraints and preprocessing measures that may be applied.

The dataset continues to expand with the WebUI2Code-12288 Version, which is characterized by a token threshold of 12,288. With 2,170 samples, this version showcases a broader range of web designs, representing the diversity of web content as the token limit grows.

Lastly, the WebUI2Code-16384 Version presents a more comprehensive collection of web designs, accommodating a token threshold of 16,384. Consisting of 2,355 samples, this version has been crafted to include websites with more extensive content and structure.

Each of these versions represents a different aspect of web design, ranging from simpler web pages to more intricate and content-rich designs. As the token threshold increases, the datasets not only expand in sample size but also in the depth and variety of content, providing a progressive method for evaluating models.

HTML Bootstrap Synthetic Dataset(s): The HTML Bootstrap Synthetic Dataset, as detailed in Section 2.2.2, establishes a more intricate foundation for testing and assessing recent encoder-decoder architectures. This dataset encompasses various design components like cards, placeholders, tables, navigation, carousels, forms, and more. Web pages generated synthetically are crafted using a collection of predefined templates that replicate real-world web design standards. These templates are carefully selected to encompass a broad range of web page layouts. By incorporating such diversity, the Synthetic Dataset becomes a valuable asset for comparing the efficacy of algorithms meant for interpreting and producing web code from visual inputs. This dataset comprises 50,000 examples, each comprising a website screenshot alongside its HTML-Bootstrap code. An alternative version of this dataset was developed featuring sketches of website elements in place of the actual rendered components. This was achieved by identifying the space occupied by each real element and substituting it with a sketched representation, as elaborated in Section

2.2.3. The outcome is a dataset containing nearly 10,000 websites (9,789), with HTML codes generated in the same manner and with identical parameters as the original Synthetic Bootstrap dataset.

Pix2code Dataset: The dataset created by Pix2Code [8] is relatively simple in its composition but serves as the baseline for comparing with existing literature. It consists of screenshots of different web and mobile interfaces, each accompanied by a Domain Specific Language (DSL) code. This DSL serves as an intermediate language that describes the structure and components of the UI, facilitating its conversion into functional codes like HTML or Android XML. The dataset is categorized into Android, iOS, and Web interfaces. The web subset, which is the main focus here, contains 1,742 samples. One notable aspect of this dataset is its simplicity. It includes only 12 distinct structural elements, with UI codes mostly composed of these elements and specific arrangement indicators. Elements like "small-title," "text," and "quadruple" are commonly found, making up almost half of all elements in the dataset. On average, a website's code in the dataset consists of 8 to 56 elements. The conversion of DSL codes into HTML resulted in a modified version of the dataset, pairing HTML codes with the screenshots. Discrepancies in text can be observed between the original screenshots and the compiled ones due to the non-deterministic nature of character generation during compilation. To maintain consistency, a version of the dataset was created that uses "lorem ipsum" as a placeholder text. In the past, the Pix2Code dataset has been used in various studies to improve certain methodologies. However, its potential as a benchmark for future advancements may be reaching its limits.

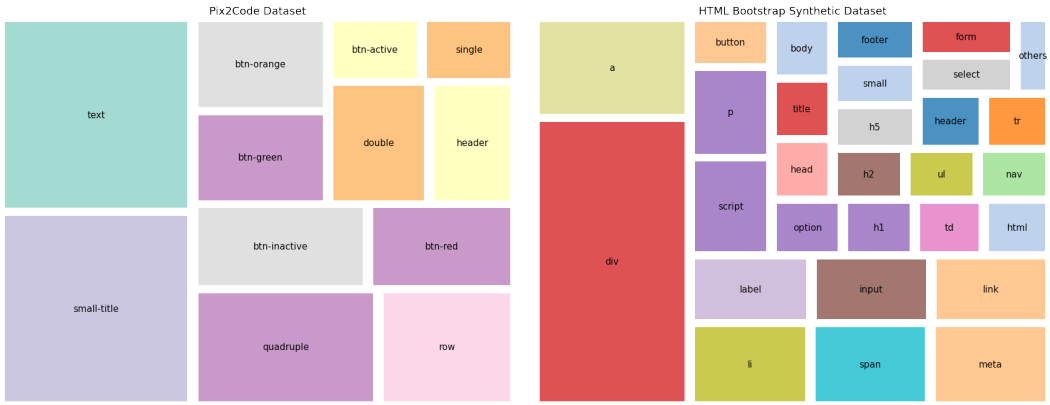


Fig. 2.6 Treeplot comparison between the distribution of tags between a) Pix2Code dataset and b) the HTML Bootstrap Synthetic Dataset. The area of the rectangles represent the distribution of the components in the dataset.

Metrics Various metrics were employed to evaluate a model's ability to predict website code from screenshots. Textual metrics predominantly utilized the original responses and predictions, while others, such as structural BLEU and HTML tree edit distance, required full code compilation and post-processing. Errors identified during corrections were saved for later analysis, and codes displayed by browsers were used to generate screenshots for image similarity metrics.

- **BLEU:** A commonly used metric for assessing machine-translated text. It measures the overlap of n-grams between generated and reference texts. However, its relevance in code prediction is limited due to the syntactic nature of code. To address this limitation, a variant known as "structural BLEU" was introduced, focusing on structural code elements by excluding non-structural elements. The Natural Language Toolkit (NLTK) was used to calculate the BLEU score, incorporating a specific smoothing function to handle precision differences in shorter texts.

- **Edit Distance:** This metric calculates the Levenshtein distance between two texts, indicating the number of character changes required to make them identical. For example, transforming "rain" into "shine" involves three modifications. The NLTK was once again used to implement this metric, with all operations assigned equal costs. A normalized version, based on the maximum character count between response and prediction, was also applied.

- **HTML Tree Edit Distance:** Driven by the concept of "structural BLEU", a distance metric that emphasizes structural elements was introduced. HTML code is represented as a tree structure, with the tree edit distance calculated using the Zhang-Shasha algorithm. The algorithm identifies the minimum node modifications needed to transform one tree into another. The Beautiful Soup parser extracted HTML nodes, and a Python implementation of the Zhang-Shasha algorithm was employed. A normalized version, taking into account the maximum node count between the two trees, was also explored.
- **Structural Similarity Index (SSIM):** In contrast to other approaches, SSIM focuses on pixel inter-dependencies, providing insights into the structure of a visual scene. The scikit-image implementation of SSIM was used. One drawback is its requirement for identical image dimensions, which may necessitate resizing and potentially alter image components, impacting the SSIM index.

Configuration

All experiments were carried out on a dedicated server equipped with four NVIDIA A100 GPUs (each with 68 GB of VRAM). This setup offered the required computational capacity for large-scale transformer training on both our Pix2Struct and FerretUI-Gemma2B models. For initial testing and prototyping tasks, we also utilized smaller-scale GPU resources (NVIDIA V100s with 32 GB VRAM or T4s with 16 GB VRAM), striking a balance between computational performance and cost efficiency.

Pix2Struct. A two-phase training schedule was employed. In the first phase, training was conducted on the synthetic dataset for 20 epochs with a learning rate of 1×10^{-4}. In the second phase, the model underwent fine-tuning on the WebUI2Code dataset for an additional 10 epochs using a learning rate of 1×10^{-5}. The batch size was set to 32, with gradient accumulation over 4 mini-batches to effectively simulate a larger batch size of 128. Adafactor was used as the optimizer, and a cosine learning rate schedule was implemented, including a 5-epoch warm-up period. Additionally, gradient clipping with a maximum norm of 1.0 was applied to ensure stable training. To process high-resolution screenshots, image inputs were limited to a maximum of 1024 patches while maintaining the aspect ratio. For text decoding, a *sliding*

window strategy was utilized on the HTML tokens: each window spanned 1024 tokens, with a 256-token overlap to provide context from previously decoded chunks. Empirically, this strategy helped alleviate memory constraints and enabled the model to autoregressively assemble longer code sequences with minimal coherence loss.

FerretUI-Gemma2B. Training Gemma2B was conducted using a similar 4×A100 GPU setup, with parameters tailored for large-scale vision-language instruction tuning. The per-GPU batch size was set at 2 (yielding a total of 8 across 4 GPUs), gradient accumulation steps were set to 2, and the training ran for 10 epochs with a learning rate of 2×10^{-5}. To address potential variations in input image resolutions, Gemma2B was configured with an "anyres" setting, resizing images to 336×336 while preserving aspect ratio metadata. The architecture and script parameters, including gradient checkpointing, half-precision BF16, a warm-up ratio of 0.03, and a cosine scheduler, were fine-tuned to ensure stable training for large instruction-tuned setups. Despite Gemma2B offering a broader context in a single forward pass compared to Pix2Struct, it comes at the cost of increased computational demands and VRAM usage due to its larger parameter count.

Both models were trained using a 90:10 split for training and testing, with a small portion of the training set (approximately 10% of total samples) reserved for validation and hyperparameter tuning. Evaluation was performed every 5 epochs for Pix2Struct and at 100-step intervals for Gemma2B, capturing metrics such as cross-entropy loss and BLEU scores. These configurations allowed for an exploration of the trade-offs between memory consumption, training speed, and real-world dataset performance.

2.2.6 Results

We compared two transformer-based methods—the relatively smaller *Pix2Struct* model and the larger *FerretUI-Gemma2B*—as well as an LSTM-based approach (Pix2Code) on various datasets to evaluate the quality of code generation. Table 2.4 presents our results, showing BLEU scores, edit distances, and visual appearance metrics (SSIM).

To compare the baseline LSTM architecture against transformer approaches, we start with the Pix2Code dataset. As depicted in Table 2.4, Pix2Struct achieves

Model/Dataset	BLEU (avg) ↑	ED (avg) ↓	N.ED (avg) ↓	SSIM (avg) ↑
LSTM				
Pix2Code	0.878	4.925	0.132	0.935
Pix2Struct				
Pix2Code	0.983	4.437	0.016	**0.942**
Synthetic Bootstrap	**0.929**	**443.890**	**0.081**	0.783
Synthetic-Sketches Bootstrap	**0.825**	**1146.678**	**0.202**	0.810
WebUI2Code-4096	**0.436**	6920.180	0.714	0.547
FerretUI-Gemma2b				
Pix2Code	**0.995**	**1.196**	**0.004**	0.939
Synthetic Bootstrap	0.740	1796.909	0.226	**0.885**
Synthetic-Sketches Bootstrap	0.725	1757.678	0.242	**0.879**
WebUI2Code-4096	0.397	**5790.563**	**0.487**	**0.978**

Table 2.4 Comparison of Metrics Across Different Datasets

a BLEU score of 0.983, outperforming Pix2Code's 0.878. The normalized edit distance (N.ED) also decreases significantly from 0.132 to 0.016, indicating a much closer match between the transformer's output and the reference code. While the SSIM values are similar (0.935 vs. 0.942), Pix2Struct demonstrates a clear advantage in both lexical (BLEU) and structural (N.ED) comparisons, highlighting the benefits of attention-based models for design-to-code translation.

On the Synthetic Bootstrap and Synthetic-Sketches datasets, Pix2Struct achieves high BLEU scores (up to 0.929) while maintaining reasonable SSIM values (e.g., 0.783). For sketches, where text is replaced by visual elements, BLEU scores slightly decrease, reflecting the increased complexity of parsing purely visual inputs. Nonetheless, the model effectively captures the structural layout (as indicated by relatively low edit distances and a strong SSIM of 0.810). These findings affirm that attention-based methods can adapt to diverse input formats, ranging from plain text elements to purely visual sketches.

We also assessed FerretUI-Gemma2B, a larger vision-language architecture, which has the potential for more robust multimodal embeddings but requires higher computational resources. As depicted in Table 2.4, Gemma2B achieves slightly lower BLEU scores on Synthetic Bootstrap (0.740) compared to Pix2Struct (0.929), yet it demonstrates higher SSIM on the same dataset (0.885 vs. 0.783). This suggests that while token-level agreement may decrease, Gemma2B better preserves the

visual layout. When analyzing real-world websites from *WebUI2Code-4096*, both models achieve BLEU scores around 0.436. Nonetheless, Gemma2B exhibits a significantly higher SSIM of 0.985, whereas Pix2Struct remains at 0.547. This distinction underscores Gemma2B's advantage in replicating complex interface visuals, albeit at the cost of requiring substantially more computational resources.

Despite achieving favorable results on synthetic and sketch-based datasets, both transformer models encounter difficulties when dealing with the intricate HTML structures encountered in real-world websites. Even with sliding-window decoding to handle longer HTML sequences, the average BLEU scores remain around 0.43–0.44. Challenges include misinterpreting complex CSS usage or generating inconsistent tag hierarchies when the code deviates from typical patterns. Notably, FerretUI-Gemma2B often compensates for textual discrepancies by producing visually accurate outputs (resulting in a high SSIM), demonstrating that different models may prioritize layout fidelity over textual precision.

Our findings validate that transformer-based techniques significantly outshine the LSTM baseline on established benchmarks (Pix2Code) and demonstrate consistent performance on synthetic datasets of various complexities. Pix2Struct typically achieves higher BLEU scores, while FerretUI-Gemma2B excels in reproducing visual likeness (SSIM, ED) but requires more computational resources. Figure 2.7 displays representative samples from each dataset.

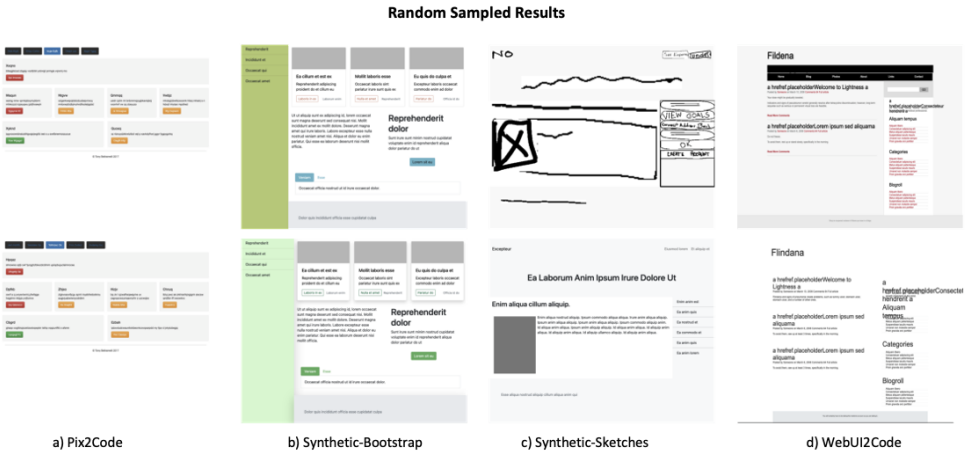


Fig. 2.7 Random samples from our data (first row) and predictions (second row).

2.2.7 Discussion

Our analysis uncovers three main discoveries concerning transformer-based methodologies for automating web design. Initially, the transformer structures vastly surpass conventional RNN-based approaches on recognized benchmarks, achieving greater precision in code creation from basic UI mockups. This confirms the potential of these structures for the design-to-code task, consistent with their triumphs in other vision-language sectors.

Subsequently, the models exhibit consistent performance on our artificial datasets, indicating their suitability for managing diverse design depictions and elements. This capacity, especially in handling both intricate mockups and sketches, shows significant promise for supporting various phases of the design process and facilitating practical design tools.

The third observation arises from our assessments on real-world websites within the WebUI2Code dataset, wherein a notable decline in performance is noted. Several factors contribute to this issue: the inherent intricacy and variability of operational websites in contrast to synthetic data; the existence of dynamic features and intricate CSS styling that may not be captured in fixed images; and the array of coding methods and optimization strategies used in real-world development, resulting in many-to-many connections between visual appearance and backend code. Furthermore, actual websites frequently incorporate elaborate responsive design structures and browser-specific optimizations that are challenging to deduce from individual images.

These conclusions hold significant implications for the evolution of automated web development. While transformer structures exhibit potential for design tools and prototype support, the disparity in performance between synthetic and real-world scenarios implies that present models may face challenges relying on real-world datasets for training.

However, utilizing models trained on synthetic data can remain beneficial as enhancement tools for tasks such as rapid prototyping, design exploration, and learning support, even in scenarios where perfect code generation is not essential. Additionally, these advancements have the potential to make web development more accessible by empowering individuals with limited coding proficiency to convert their designs into operational code. The sketch-to-code capabilities in particular can enhance rapid prototyping and early-stage design exploration. Nonetheless, the

effective implementation of these technologies will rely on continual enhancements in model architectures and training data, as well as their strategic integration into existing design workflows and tools to maximize their advantages for end users.

Implications for Engineering of Interactive Systems

The conversion of UI designs into operational prototypes using automated code generation shows great potential for both beginners and experts in interactive systems. For end users, the capability to draw a layout and receive a functional model significantly reduces the obstacles to technology development. This strategy is in line with the overarching objectives of user-centered design, emphasizing empowerment and simplicity of refinement. Instead of needing in-depth understanding of technical details, users can quickly experiment, improve, and validate their concepts in a tangible format. This increased accessibility could encourage broader involvement in UI design, nurturing collaborative communities where design concepts can flow more openly.

On the other hand, experienced developers and designers often face constraints due to deadlines and the need to assess various design options. By implementing a design-to-code process utilizing transformer architectures, these professionals can delegate the more routine or mechanical tasks involved in prototype development. The system's acquired ability to produce logical code structures from abstract ideas expedites the overall workflow. In conventional development cycles, minor adjustments to the interface design could trigger a series of manual coding and layout modifications; in this scenario, the alterations are managed at a conceptual level, with adjustment mechanisms handling the details of low-level coding. These improvements in efficiency not only offer a quicker path to refined prototypes but also allow human expertise to focus on impactful creative and user-oriented activities.

Moreover, these techniques establish potential routes for seamless integration into contemporary design workflows. Within numerous teams, interactive design initially takes shape within specialized software—ranging from wireframing tools to more tailored prototyping platforms—prior to transitioning into code. The concept of connecting these tools with a robust transformer model implies that the entire process from idea conception to interactive demonstrator creation can be significantly expedited. Furthermore, the iterative nature of user testing, a crucial aspect of research in human-computer interaction, is also positively impacted by this efficient

approach: due to the rapid generation and modification of prototypes, user feedback loops can occur more frequently, facilitating real-time design improvements. Ultimately, the combination of user-centered design principles with sophisticated language-modeling techniques has the potential to transform the way interactive systems are developed, making them more accessible to newcomers and increasing efficiency for experienced professionals.

Adaptive Learning and Personalization

As AI systems increasingly influence web design and development, they are moving beyond static, one-size-fits-all outputs. Instead, emerging approaches can adjust to individual user behavior patterns, providing personalized code-generation experiences. By examining a designer's previous decisions, a modern AI platform can learn to predict future preferences and propose UI components or design layouts that match the user's evolving style. This adaptability is based on sophisticated algorithms: reinforcement learning enables a system to continuously modify its parameters according to real-time feedback, while transfer learning allows knowledge obtained in one domain—such as layout heuristics—to aid in design tasks in another. From a perspective of human-computer interaction, these adaptive AI systems intersect with both End-User Development (EUD) and Adaptive User Interfaces (AUIs). Through EUD, non-technical individuals can guide the AI to concentrate on specific design constraints or branding elements without formal coding skills. On the other hand, AUIs adjust their interactions dynamically based on user feedback, catering to various skill levels and interface preferences. These aspects collectively support inclusivity and empower users, ensuring that automated design tools are responsive to each designer's artistic direction. Our contribution establishes a basis for these adaptive workflows by illustrating how an AI design-to-code system can learn not only general design patterns, but also context-specific signals gathered from repeated user interactions. As time progresses, the model's predictions become more personalized, facilitating more efficient prototyping and smoother creative exploration. This concept of adaptive design assistance propels the field towards intuitive, user-centered methodologies, bridging the gap between automated code generation and designer-led innovation.

2.2.8 Summary

Our study makes significant contributions to the advancement of web development in several ways. The introduction of an enhanced WebUI2Code dataset, particularly its data collection process, establishes a solid groundwork for converting mock-ups into code solutions. Additionally, our synthetic datasets facilitate the exploration of sketch-to-code systems, making it easier for non-technical individuals to participate in web development. Our evaluation of two recent transformer-based models shows enhancements in traditional benchmarks and successful outcomes with our synthetic data. These findings are in line with current trends in artificial intelligence, indicating a future where intricate and diverse web designs can be accurately transformed into functional websites. Moving forward, our focus will be on gathering even larger datasets and conducting extensive training on a large scale. This will be followed by thorough experimentation involving designers and actual users. We foresee the development of systems capable of learning and adapting to individual design preferences and workflows. These systems could utilize methods such as reinforcement learning to adjust based on user input and transfer learning to handle a variety of design scenarios, resulting in more personalized and responsive design experiences. By integrating these technological advancements with practical workflows and needs, we aim to create precise and user-centered design enhancement tools. Our contributions are geared towards promoting web development that follows human-centered design principles and fosters creativity. Ultimately, our objective is to enhance human creativity and productivity through AI systems.

2.3 Style-Aware Sketch-to-Code Conversion

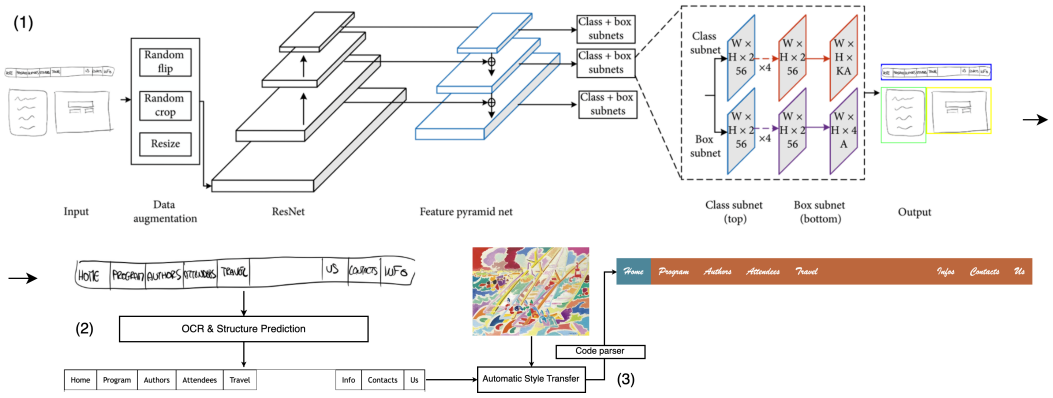


Fig. 2.8 Overview of the method. (1) Commencing from the sketch of a webpage, a segmentation of its interface is conducted. (2) The structure and textual elements of the chosen component are deduced (Section 2.3.1). (3) Style characteristics of the reference image are extracted and incorporated into the structural features of the sketch. Subsequently, a parser generates the final code alongside the rendering of the component (Sections 2.3.2 and 2.3.3).

The process of converting a sketched interface into computer code involves three main sub-problems. Initially, it involves understanding the sketch and identifying the existing elements along with their positions. Next, each identified component needs to be styled according to a specific reference style. Lastly, the final challenge is to produce the code for the styled component. The methodology described was developed using Python 3.8. PyTorch was utilized for implementing the neural network, PIL and OpenCV for image processing, and Pandas and Numpy for data processing.

2.3.1 Understanding of Sketches

The process of comprehending the sketch is a task in computer vision that involves detecting and identifying the components (e.g., buttons, navbars, etc.) and their relative positions within the sketch of a Web-based GUI. In this task, we have employed the approach used in Sketch2Code [9], which leverages RetinaNet [56], a well-known single-stage detector known for its accuracy and speed. RetinaNet utilizes a feature pyramid network to efficiently detect objects at various scales and introduces a novel loss function, the focal loss, to address the issue of extreme

foreground-background class imbalance. This model can predict both the class and the positional box of the object being detected simultaneously. The same dataset as in Sketch2Code is used for this task. Additionally, an OCR technique is utilized for recognizing written words in the sketch.

2.3.2 Automatic Style Transfer

Considering that a Web component comprises various visual elements, we divide the task of extracting the style from the chosen reference image into two sub-problems: *color extraction* and *text style selection*.

Color Extraction. The most prominent colors from the selected reference image are identified using a median cut-based clustering approach. This technique involves organizing data with multiple dimensions into sets by iteratively dividing each data set at the median point along the longest dimension, particularly the color dimensions in the image (i.e., the RGB channels in a colored image).

Text Style Selection. To determine the text style that best matches the selected reference image, we leverage the feature extraction capabilities of Convolutional Neural Networks (CNNs). Specifically, with a list of preferred fonts, both the image and a sample sentence for each font are fed through a pretrained Visual Geometry Group (VGG) neural network. The font that minimizes the cosine similarity between the two hidden representations is then selected. In mathematical terms, for a model S, a reference image $i \in I$, and a set of fonts F, the j-th style $f \in F$ is chosen such that

$$j = \arg\min_{j} S(i) \times S(f_j) \tag{2.1}$$

where $S(i)$ and $S(f_j)$ represent the final layer activation values of the network. The above equation guarantees that the chosen reference image and the font share some visual similarity, or at least that the similarity is maximized among the selected fonts.

2.3.3 Code Generation and UI Reconstruction

After obtaining the customized component from the previous step, it is processed once more through a multi-headed VGG Convolutional Neural Network [57] to analyze its characteristics, both in terms of structure and style. The outcome is then forwarded to an external parser, along with the corresponding box positions derived from the sketch understanding phase (Section 2.3.1), in order to generate the ultimate code representation of the GUI. The external parser correlates the features deduced from the CNN with CSS attributes for style characteristics and HTML attributes for structural components. Eventually, an algorithm integrates them into a website template to produce the final interface.

2.3.4 Preliminary Results

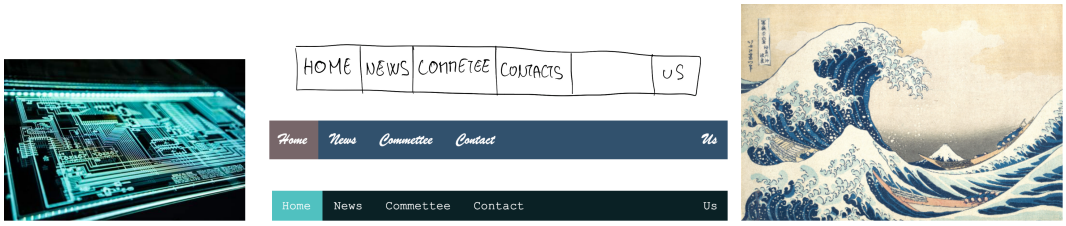


Fig. 2.9 Results of the automatic style transfer algorithm and the UI reconstruction. The sketch above is transformed into the respective navbars with styles matching the images on the sides.

In order to validate our approach, our experiments primarily focus on verifying the accurate prediction of the structure of the sketch using both synthetic and real sketches, obtaining correct visual feedback from the extraction of dominant colors from the reference image, and confirming the robustness of the selected text font that closely matches the style of the reference image across different sentence samples. As the segmentation and reconstruction techniques were inherited from and previously validated by Sketch2Code [9] and UICode [58], we do not discuss their performances. The experiments were carried out using Google Colab.

Dataset To assess the effectiveness of the style transfer method, we constructed a synthetic dataset of 3,000 navigation bars (navbars) sketches. These navbars can contain a maximum of 5 items floating left and 3 items floating right, and the goal

of the structure prediction model is to determine the number of right and left items. While this focus on a single component type limits the diversity of the dataset, it provided a controlled environment to validate the core style transfer mechanism before scaling to more complex layouts. Additionally, we fine-tuned the resultant model using 30 real navbar sketches to evaluate its performance in a real-world scenario.

Measures The efficacy of our proposed method is based on three primary objectives.

1. It is necessary for the convolutional neural network to accurately classify the structural attributes of the sketched component for subsequent parsing into code.
2. The color extraction algorithm should provide accurate visual feedback on its ability to extract the correct colors and select the appropriate font based on the reference image.

'When it comes to the first objective, the classification performances of the CNN on both the synthetic and real sketches dataset are assessed. This involves the network correctly inferring the content features of a given sketch. The evaluation of its performance is based on the accuracy of the prediction compared to the ground truth. Regarding the second objective, the performances of the color extraction algorithm are visually evaluated based on the obtained results. A more comprehensive evaluation involving designers is left for future work.

Experiments To assess the performance of the CNN for the *sketch structure prediction*, the synthetic sketch dataset was divided into 2,500 training samples and 500 test samples. The network was trained for 20, 30, and 50 epochs using pretrained weights from ImageNet [59], followed by fine-tuning on 50 real sketches and testing on 20 of them.

As shown in Table 2.5, the convolutional network's performance in distinguishing the structural features of the sketched component achieved very good results with a top 0.732 accuracy over the real sketches set after 50 epochs of training. The dominant *color extraction* algorithm's performance has been visually assessed on 10 reference images, demonstrating qualitative optimal results.

Epochs	Synthetic Sketches	Real Sketches
20	0.898	0.701
30	0.909	0.725
50	0.912	0.732

Table 2.5 Accuracy in predicting the structural characteristics of the synthetic and real sketches

2.3.5 Summary and Future Work

This chapter introduces an approach to assist designers in creating webpages from a sketch while incorporating style elements. The method comprises three key components: a deep learning architecture for segmentation and classification, a style extraction process from a reference image, and a parsing algorithm. One of its strengths is its full integration and ease of adaptation to various sketch types across different domains. Notably, it represents the first model known to enable designers to automatically customize the style of a sketched component using a chosen template. Furthermore, it is highly modular, allowing for individual module changes without necessitating modifications to preceding components.

Our methodology does have certain limitations that could be addressed in future studies. Firstly, the style and content characteristics of components are manually crafted, restricting the model's ability to generalize beyond the initial sketch specifications. This approach was chosen to achieve optimal performance given the intricate nature of style specifications in web components. A potential avenue for future research involves implementing techniques that facilitate visual style transfer using language models instead of procedural methods, as language models have demonstrated the capacity to generalize beyond manually crafted features in this context, as evidenced by Beltramelli [8] and Chen et al. [58]. Secondly, the current automatic style transfer method is confined to predefined stylistic attributes, such as colors and fonts. In real-world web design scenarios, numerous other stylistic attributes need to be considered, including shadows, borders, and the responsiveness of elements. Addressing these complexities to enable the widespread application of the proposed method in practical settings may present challenges. Therefore, future research efforts should concentrate on developing automatic models capable of managing such intricate stylistic considerations in a comprehensive manner.

Finally, the suggested approach must undergo testing with a varied range of sketches and web components, and be integrated into a design tool that enables designers to choose different styles for their sketches. Subsequently, this tool will be subject to evaluation through user studies to determine the effectiveness of the overall strategy.

In conclusion, the translation of user interface sketches into code is approaching implementation in practical applications, and our research represents an initial effort to enable automatic stylization of GUI elements through the utilization of machine learning methods, aiming to provide a more comprehensive approach to support designers in streamlining this laborious aspect of their work.

2.4 Dynamic Behavior Specification in Sketches

In the comparative study by Silva et al. [60], the concept of *behavior specification* is defined as the capability to incorporate dynamic behaviors into prototypes. "Behavior" is characterized as a collection of states that prototypes can transition between. While most prototyping tools primarily support the creation of static mock-ups, only a few model the dynamic behavior of the prototype.

As outlined in [60], the primary approaches for specifying the prototype's behavior involve setting hotspots on images and managing events on widgets. Hotspots refer to highlighted areas overlaid on the prototype sketch to capture user-triggered events[1]. Designers typically need to establish one hotspot for each interactive part of the interface. However, a limitation of this method is that hotspots are linked to graphical regions based solely on their coordinates, rather than semantically connected to the graphical elements depicted in the image.

Wireframe tools utilize widgets for constructing the interface[2]. These event handlers typically define the action needed to trigger an event and the subsequent behavior it initiates.

Examples of tools that support wireframe interactions include Balsamiq[3], ActiveStory Enhanced [61], SILK [62], and DENIM [63]. Additionally, tools like

[1] https://marvelapp.com/
[2] https://pidoco.com/en
[3] https://balsamiq.com

AppSketcher[4] and JustInMind[5] provide capabilities for specifying conditions, modifying properties, and utilizing variables; Appery.io[6] and ScreenArchitect[7] also offer the ability to program code.

Our methodology characterizes the dynamic nature of the GUI from the sketch itself by employing a specific set of symbols. It enables wireframe interactions without requiring the addition of widgets or hotspots at a later phase. Additionally, as far as we are aware, the suggested technique represents the initial effort to directly model behavior specifications from a sketch via convolutional neural networks.

2.4.1 Method

This section introduces a technique for automatically creating code from sketches, as well as a new set of symbols and the corresponding process for representing dynamic behavior.

2.4.2 Modeling Dynamic Behavior

In order to model the dynamic behavior of the prototype, a group of symbols representing various dynamic behaviors is introduced. These symbols should be depicted directly on the sketch and are selected based on the essential dynamic characteristics identified in the literature, i.e., from [60, 62]. The following group has been chosen to showcase the feasibility of the model and will be further developed in upcoming research to encompass a broader spectrum of dynamic behaviors:

Default Selected Element denotes the item that is preselected in the sketched interface. An example of such an item is the “Home” button in a horizontal navigation bar of a web application.

Dropdown Menu signifies that the element triggers a dropdown menu.

Page Indicator distinctively identifies the sketched interface, e.g., the page destination of a link.

[4] https://www.uxplaza.com/appsketcher
[5] https://www.justinmind.com
[6] https://appery.io
[7] https://www.screenarchitect.com

Link symbolizes the association between the sketched element and a specified page.

Figure 2.11 presents the four symbols that were introduced, along with their respective functions, the reasons for selecting the representations, their application, and an illustration for each symbol. The figure may enhance comprehension of how the symbols are incorporated into an actual sketch of a GUI.

2.4.3 Generation of Prototype Interface

The process of creating the code for an interface from a sketch can be divided into three main sub-problems.

Initially, the challenge involves segmenting the sketch based on semantic elements, such as navbar, list, carousel (Section 2.4.3). Subsequently, for each semantic component provided, the task is to analyze the structural characteristics of the sketch and determine the existing symbolic elements and their positions (Section 2.4.3). Lastly, the final obstacle is to produce the code for the resulting component, considering the dynamic behavior expressed (Section 2.4.3).

The approach described has been implemented using Python 3.8. The neural network was developed utilizing PyTorch, image processing with PIL and OpenCV, and data processing carried out using Pandas and Numpy.

Interpretation of Sketches

Interpreting a sketch is a computer vision task that involves detecting and recognizing the components included in a Web-based interface sketch (e.g., buttons, navbars, etc.) and determining their spatial relationships.

For this purpose, we applied a similar method as Sketch2Code [64], which utilizes RetinaNet [65], a widely-used single-stage detector known for its accuracy and efficiency. RetinaNet is capable of predicting both the class and position of the object being detected simultaneously. The network architecture is illustrated in Figure 2.10 (1).

Understanding Components

After segmenting the user interface sketch, we predict the structural characteristics of the components and identify symbols to represent dynamic behavior.

In particular, as illustrated in Figure 2.10 (2), we employ a convolutional neural network, customized for each element, trained to categorize the structural characteristics of the sketched element, for example, in the scenario of a navbar, the quantity of elements floating left and right. We utilize the identical network to forecast which (*symbol type*) and where (*in which element*) symbols exist. To establish connections between multiple pages, we utilized the page indicator, i.e., a distinct number, inscribed on the top right of the sketched page.

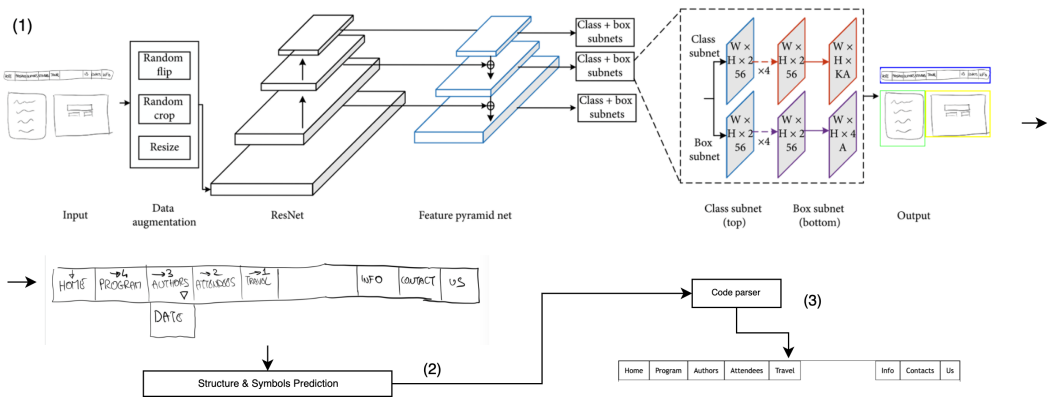


Fig. 2.10 Overview of the method. Starting with one or multiple sketches of interfaces, in (1) a segmentation of the single sketch into sub-components is performed. Then, for each component, a convolutional neural network is used to infer its structural and dynamic properties (2). Finally, with the assistance of a parser, the predicted properties are translated into the backbone code of the sketched component. (3) depicts the rendered Web element resulting from the entire process.

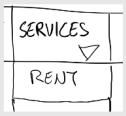
SYMBOL	FUNCTION	MOTIVATION	USAGE	EXAMPLE
	Default selected element.	The symbol has been chosen for its visual similarity with an anchor.	The symbol must be drawn upon the element to be selected by default.	
	Dropdown menu	The symbol has been chosen because it is the standard representation of the dropdown menu in literature.	The symbol must be drawn below the element that activates the menu	
	Page Indicator	NOTE: the symbol can be any number.	The number must be written inside a square in the upper left of the sketched page that it uniquely identifies	
	Link	The symbol has been chosen for its visual meaning of motion.	The symbol must be drawn above the button and should be followed by the page unique number to which the link points to.	

Fig. 2.11 The proposed set of symbols to model dynamic behavior in sketch prototypes: The *default selected element* symbol is used to model the item that is selected by default in the given interface; the *dropdown menu* symbol indicates that the element opens a dropdown menu; the *page indicator* symbol is used to link together different page sketched by the designer; the *link* symbols, links a sketched element into an indicated page.

Code Generation

After considering the structural properties of the component, as well as the type and positioning of symbols used to represent the dynamic behavior, we then move on to creating the code for the backbone using a parsing function. Figure 2.10 (3) illustrates the final rendered component.

2.4.4 Experiments

We aim to validate that our approach accurately produces the code for the navbar with dynamic behavior, assess the structural characteristics of the sketch, and identify symbols representing dynamic behavior correctly. Due to the absence of a dataset of hand-sketched web interfaces, we initially train our method using a synthetic dataset and subsequently fine-tune it using a set of 50 real sketched navigation bars (navbars). Since the segmentation and reconstruction techniques were previously validated in Sketch2Code [66] and UICode [67], we do not provide their performance evaluation.

Dataset To evaluate the efficacy of our method, we created a synthetic dataset consisting of 3,000 sketches of navbars. Each navbar may contain a maximum of five items aligned to the left and three items aligned to the right. The symbols used to represent dynamic behaviors included a default selected element, multiple links, multiple dropdown menus, and a unique page indicator. The goal of the structural prediction model is to predict the number of items on the left and right. This deliberate focus on a single, common component allowed for a controlled evaluation of the symbol recognition and behavior generation logic. Additionally, we fine-tuned the model using a set of 500 real sketched navbars to assess its performance in a more realistic setting. The real sketches dataset exhibits greater variability compared to the synthetic dataset, featuring hand-drawn lines, overlapping elements, and mispositioned components.

Measures The Convolutional Neural Network (CNN) should accurately classify the structural attributes and the type and position of symbols within the sketched component to facilitate conversion into code. We use *accuracy* as the primary performance metric.

Experiments To assess the effectiveness of the CNN in the *sketch structure prediction* task, the synthetic sketch dataset was divided into 2,500 training samples and 500 testing samples. The network was trained for 20, 30, and 50 epochs using pre-trained weights from ImageNet [68], followed by fine-tuning on 400 real sketches and testing on 100 sketches.

Epochs	Synthetic Sketches	Real Sketches
20	0.991	0.968
30	0.995	0.973
50	0.998	0.982

Table 2.6 Accuracy Results over Synthetic and Real Sketches Datasets

According to the data presented in Table 2.6, the convolutional network's ability to differentiate the structural characteristics of the sketched component yields excellent outcomes, achieving a top accuracy of 0.982 on the actual sketch dataset after 50 training epochs. This indicates the network's proficiency in identifying the sketched

component's structure, which can then be translated directly into code, showcasing the potential of this approach for practical applications.

Dataset	Default Selected	Dropdown Menu	Link	Page Number
Synthetic Sketches	0.995	0.996	0.989	0.992
Real Sketches	0.987	0.991	0.972	0.989

Table 2.7 Accuracy Results for each symbol

Moreover, for a deeper insight into the numerical outcomes of our approach, the network's performance in identifying each of the proposed symbols is examined in Table 2.7. This analysis could serve as a basis for potential modifications in the designs of certain symbols to enhance accuracy.

According to the findings presented in Table 2.7, the symbol "Page Number" stands out as the most readily identifiable by the classifier, whereas "Dropdown Menu" and "Default Selected Elements" exhibit similar performance. Conversely, "Links" is the symbol with the lowest recognition rate, likely due to occasional overlap with text in some instances. Future efforts may involve refining its design or positional specifications to yield improved outcomes.

2.4.5 Summary and Future Work

In this chapter, a technique is introduced to aid designers in creating web pages from sketches, while illustrating the dynamic behaviors of these pages. The method comprises four key components: a predefined set of symbols for sketching web pages; a deep learning framework for segmenting sketched pages into components; a classification model for deducing the structural attributes of the components and identifying symbols representing dynamic behaviors; and a parsing model that leverages the network's output to produce the final code. Notably, this method is fully integrated and easily modifiable for a variety of sketches in different fields. To the best of our knowledge, it is the initial approach enabling designers to capture the dynamic behavior of sketches within sketch-to-code translation algorithms by utilizing deep learning methodologies.

While the proposed technique offers significant advantages, there are certain limitations that could be resolved in future studies. Particularly, the structural

attributes of the components are manually crafted, restricting the model's ability to generalize beyond the sketching specifics. This design choice is deliberate to achieve accurate outcomes owing to the intricate nature of structural specifications in web components. Subsequent research will focus on incorporating methods enabling code generation using language models instead of procedural techniques, as language models can generalize beyond manually crafted features in this context, as demonstrated by Beltramelli [8].

Secondly, the representation of dynamic behavior is constrained to a subset of the dynamic behavior of an authentic web application. Subsequent investigations should concentrate on improving the capabilities of our technique to represent a broader spectrum of dynamic behaviors. Finally, the illustrated technique should undergo testing with a varied assortment of sketches, hand-drawn symbols, and web components, and should be integrated into a designer tool. Such a tool will then be assessed in user studies to evaluate the effectiveness of the overall approach.

In conclusion, the conversion of user interface sketches into code is moving closer to implementation in practical applications, and our work takes initial steps towards enabling designers to model the dynamic behavior of web interface elements. This is achieved by utilizing machine learning methods to provide a more comprehensive approach that can aid designers in simplifying this laborious aspect of their work.

2.5 Conclusion

Our investigations into design-to-code translation reveal both significant advancements and inherent limitations in current approaches. The transformer-based architectures substantially outperform traditional LSTM models in controlled environments, with both Pix2Struct and FerretUI-Gemma2B achieving near-perfect BLEU scores on benchmark datasets. However, performance degrades markedly on real-world interfaces—a discrepancy that highlights the fundamental gap between synthetic training environments and the complex multivariate relationship between visual design and implementation code.

The style-aware and behavior-specification extensions address critical dimensions overlooked in previous research. Style transfer through reference images enables the preservation of designers' aesthetic intentions without requiring explicit

coding knowledge, while symbolic annotations for dynamic behaviors bridge the longstanding divide between static mock-ups and interactive prototypes. These approaches fundamentally reposition the sketch-to-code pipeline from a structural translation tool to a more complete design expression medium.

This work exposes a critical tension in automated interface generation: between the deterministic nature of code generation and the inherently under-constrained nature of design. A single visual representation can map to numerous valid implementation patterns, each with different implications for performance, maintainability, and browser compatibility. Future systems must therefore move beyond simple input-output mapping to consider the sociotechnical context in which designs exist—incorporating awareness of design patterns, accessibility requirements, and cross-platform considerations.

The persistent performance gap on real-world websites suggests that hybrid approaches may prove most effective in practice. Rather than attempting complete automation, future systems might serve as augmentative tools that accelerate specific aspects of the implementation process while preserving human oversight for critical design decisions. This partnership model aligns with the observed strengths of our approach in capturing structural and stylistic elements while still struggling with complex responsive behaviors and cross-browser optimizations.

As design tooling evolves, the boundaries between design and implementation will continue to blur. Our contributions provide mechanisms for expressing design intent more comprehensively in early sketches, potentially reshaping development workflows to emphasize iterative refinement over sequential handoffs. The true value of these approaches lies not merely in automation efficiency but in enabling more fluid expression of creative intent across the traditionally siloed domains of design and development.

It is also important to situate this work in its historical context. The research presented in this chapter was conducted prior to the widespread availability and viability of large-scale generative models (LLMs) for such tasks. Consequently, the methods relied on established neural architectures (e.g., LSTMs, early transformers) that were considered state-of-the-art at the time. The landscape has since shifted dramatically with the emergence of powerful commercial and open-source GenAI tools, many of which now offer sophisticated prototyping and code generation capabilities that build upon the foundational concepts explored here. Acknowledging

this rapid evolution is crucial for understanding both the contributions and the limitations of this earlier work.

Chapter 3

Empowering Educators: AI-Assisted Tools for Instructional Design

Publication notice. Early versions of the approaches presented in this chapter appeared in *ACM Learning@Scale'24* ("Towards Educator-Driven Tutor Authoring: Generative AI Approaches for Creating Intelligent Tutor Interfaces") and are currently being extended in an article under submission.

3.1 Introduction

Intelligent Tutoring Systems (ITSs) provide personalized and adaptive education by offering practice problems tailored to students' expertise levels along with appropriate feedback during problem-solving. Research by Bloom et al. [24] demonstrated that students receiving one-on-one tutoring achieve significantly better learning outcomes compared to those in traditional lecture-based instruction. By tracking individual learner knowledge and skills, ITSs can customize problem sequences to improve learning efficiency, enabling tutors to serve many students simultaneously and address supplemental instruction needs in large classes [69]. Multiple studies confirm that ITSs effectively enhance learning outcomes [25].

A key pedagogical approach employed by effective ITSs is *scaffolding*, which encourages students to extend their reasoning processes, resulting in higher task completion rates compared to conventional answer-based systems lacking supportive

features [70]. These effective pedagogical techniques frequently lead to students successfully completing learning tasks, while systems based solely on final answers tend to cause students to abandon tasks midway through the learning process [70]. Several meta-analyses indicate that certain ITSs are more effective in supporting student learning than other computer-assisted instruction systems or traditional human teaching in large classes [71, 72, 70].

Despite their proven effectiveness, widespread adoption of ITSs has been hindered by the complexity of their development, which typically requires specialized programming and design expertise [26]. This complexity prevents non-technical educators from creating customized tutors [28], limiting development to technical experts like researchers and software developers. The creation of effective ITSs involves not only domain knowledge but also interface design skills that most educators lack.

To democratize the creation of ITSs and empower educators in their development, various authoring tools and platforms have emerged. Notable examples include the Cognitive Tutor Authoring Tools (CTAT) [28], which enable tutor creation through demonstration of problem-solving steps and specification of pedagogical rules; the Authoring Software Platform for Intelligent Resources in Education (ASPIRE) [73], which allows domain experts to develop constraint-based tutors; and machine learning approaches like SimStudent [74, 75] and the Apprentice Learner System [76], which have been applied to model student learning and support tutor authoring.

While these tools help mitigate the challenges of tutor model creation, they predominantly focus on developing the domain model and pedagogical strategies while assuming educators possess the necessary interface design skills. The Apprentice Tutor Builder (ATB) [29], for instance, offers a drag-and-drop interface for assembling tutor layouts in row and column formats, enabling educators to iteratively develop tutoring models without programming. However, ATB still implicitly assumes that educators can effectively design interfaces without specific training or support. Research demonstrates that interface and instructional design requires specialized skills [27] that most educators lack.

Recent work by Calò and Maclellan [77] began to address this gap by using generative AI to translate teaching requirements directly into ITS interfaces. Their approach leverages prompt engineering and domain-specific language to generate interfaces based on educators' specifications. While their method shows promis-

ing time reductions compared to drag-and-drop builders, it lacks integration with established ITS practices, relies heavily on prompt specification (challenging for non-expert users [78]), and has not been validated with actual educators. The generation process occurs without educator supervision, exposing the process to the limitations of prompt writing.

The recent emergence of generative AI applications has created new possibilities for interface development support across domains. Large language models (LLMs) have been used to interact with UIs through natural language [79], generate CSS properties based on design objectives [23], and create functional UI prototypes through prompted responses [80]. These developments align with Nielsen Norman Group's concept of **outcome-oriented design**, where users focus more on setting parameters and constraints for AI systems than on specifying individual design details [81]. Similarly, Shneiderman's research on Human-Centered AI [82] emphasizes designing systems that leverage AI capabilities while ensuring users maintain control and understanding of the process.

The integration of generative AI with intelligent tutoring systems has mostly focused on using LLMs as substitutes for expert tutor models, providing immediate feedback, continuous availability, and enhanced flexibility compared to traditional rule-based tutors [83, 84]. LLMs have demonstrated effectiveness in solving questions across various subjects [85–88], identifying and correcting mistakes in student work [89–91], and improving knowledge tracking and content customization [92–95].

However, significant limitations have been identified in using LLMs directly as tutoring agents, particularly regarding reliability [96], user interactions [97, 98], and fairness in educational settings [99, 100]. These challenges highlight the importance of keeping educators involved in the tutoring system development process.

This chapter presents two complementary approaches that take a different direction by leveraging generative AI not as the primary tutoring model but as a design assistant that helps educators create effective tutor interfaces. First, we introduce a system that enhances the Apprentice Tutor Builder with generative AI capabilities to produce complete interfaces or specific components based on educators' requirements. This approach incorporates design constraints to ensure generated interfaces are effective and visually appealing while maintaining educator control through options for whole-interface or component-specific generation.

Building on this foundation, we then present an advanced approach that breaks down high-level teaching requirements into pedagogical steps before generating multiple interface options for preference-based refinement. By decomposing requirements into structured instructional actions (e.g., concept introduction, guided practice, feedback loops) derived from established ITS strategies [70], this approach ensures alignment with effective pedagogical practices while allowing educators to maintain control over the tutoring process. Drawing inspiration from DirectGPT [101], we introduce a preference-driven UI refinement method that enables educators to select preferred elements from diverse AI-generated drafts, guiding the design process without requiring expertise in UI design or complex prompt engineering.

This approach aligns with the UI design exploration-exploitation process [102], balancing creativity and discovery with optimization. During the exploration phase, designers explore a wide range of options, while in the exploitation phase, they refine the most promising candidates. Our AI-assisted tool makes this typically time-consuming and expertise-requiring process accessible to educators without technical background, enabling them to engage in structured, iterative design guided by their pedagogical expertise.

Through preliminary evaluations and a user study with eight K-12 educators, we demonstrate that these approaches significantly enhance both interface quality and user satisfaction compared to traditional methods, without extending design time. The implications of our work extend beyond intelligent tutoring systems to the broader field of end-user development, potentially accelerating the democratization of AI-assisted design tools in education and enhancing the accessibility of personalized learning technologies.

3.2 Generative AI-Enhanced Tutor Builder for Rapid Interface Prototyping

Our approach to enhancing the ATB with generative AI capabilities comprises three main components: a Domain Specific Language (DSL) for interacting with the LLM, prompt engineering to direct the generation process, and two tiers of interaction for adaptability and supervision.

We introduce a concise DSL for illustrating tutor interface structures, encompassing basic elements like `title[value]` for specifying the tutor's title, `row` and `column` for indicating horizontal and vertical layouts of elements, `label[value]` for defining text labels, and `input[placeholder]` for specifying input fields with optional placeholder text. These components can be combined to create intricate tutors. The DSL representation facilitates effective communication with the LLM and guarantees that the resulting HTML output complies with the desired template, thereby avoiding inconsistencies and deviations from the intended design and aesthetics that could occur if a tutor's HTML interface were directly generated from the LLM.

To assist the generative model in developing suitable tutor interfaces, we constructed a prompt comprising distinct sections. The **System Description** outlines the desired tutor interface, stressing the significance of a clear problem statement and a step-by-step resolution pathway to align with educational goals. The **Format Explanation** elucidates the DSL format utilized for representing the tutor interface layout, enabling the model to generate layouts that adhere to the specified format. The **Design Instructions** detail design principles, such as segregating input elements, organizing elements in rows and columns, and refraining from incorporating interactive buttons in the layout. The **Task Instruction** directs the model to convert a comprehensive tutor description into a concise DSL representation, ensuring comprehension of its primary goal and generation of the intended output format. Lastly, a collection of illustrative **Examples** functions as a guide for the model on implementing the aforementioned principles in diverse scenarios.

We present two levels of interface generation tailored for different stages of design. Initially, **Interface Generation** facilitates the creation of a complete tutor interface layout based on a detailed description, serving as the foundation for tutor design. This enables users to commence from a generated user interface that adheres to the design principles outlined in the prompt. We posit that a refined user interface is more likely to adhere to these principles. Moreover, interface generation may prove particularly beneficial for users seeking an automated approach or facing time constraints. Conversely, **Component Generation** produces designs for specific and reusable interface elements, such as widgets for equations or other data input formats. This feature allows users to develop interfaces that closely align with their specific objectives on a smaller scale, while still capitalizing on the efficiency and consistency offered by the generative model.

Example Workflow To illustrate, consider an educator designing a simple algebra tutor. They might provide the high-level requirement: "Create a tutor to solve the linear equation 2x + 5 = 15. The student needs to first subtract 5 from both sides, and then divide by 2 to find x." The AI-enhanced builder would process this and, using the Interface Generation feature, produce the corresponding DSL: `title[Solving 2x + 5 = 15] column{ row{ label[Step 1: Subtract 5 from both sides] input[2x = 10] } row{ label[Step 2: Divide by 2] input[x = 5] } }`. This DSL is then instantly rendered as a clean, structured interface in the ATB, which the educator can further refine.

Our AI-enhanced tutor interface constructor is built on top of the existing ATB interface, utilizing an HTML/JavaScript front-end and a Flask backend. We employ GPT-4[1] as the language model engine for the generation process. The interface and component generator are seamlessly integrated into ATB's user interface as toolbar widgets.

Interface Type	Time (s)			Keystrokes		
	Classical	AI-Enhanced	Reduction	Classical	AI-Enhanced	Reduction
Simple	187	**143**	-23%	184	**126**	-31%
Complex	372	**116**	-68%	141	**74**	-47%

Table 3.1 A Comparison of the Time and Keystrokes Needed for Constructing Tutor Interfaces: Classical versus AI-Enhanced

Initial Assessment

In order to assess the effectiveness of the AI-enhanced Apprentice Tutor Interface Builder, we conducted a small-scale comparison with a previous iteration. The performances of four team members were compared against those of high-expertise individuals as reported in the ATB paper. While this comparison is not strictly controlled, we consider the high level of expertise of the ATB participants to significantly impact their performance, thus making it a suitable reference point for initial comparison. Additionally, to validate our findings and address issues related to comparing diverse participant groups, we utilized the Keystroke-Level Model [103]. This model assesses interface efficiency by calculating the minimum number of

[1] version gpt-4-0613

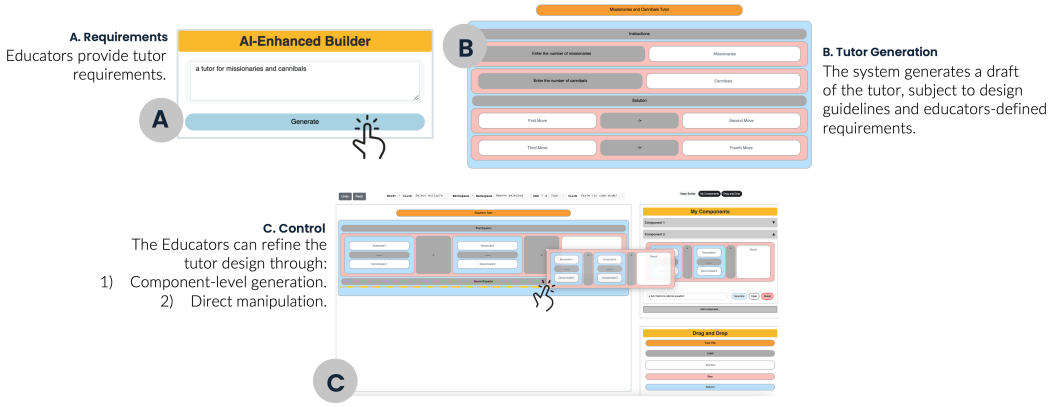


Fig. 3.1 Illustration of our AI-Enhanced Tutor Builder System: Educators provide input requirements (A) which inform the automated generation of a draft tutor interface (B), followed by the educator's hands-on refinement through component generation and direct manipulation (C), visualizing the integration of generative design and educator-driven customization.

keystrokes needed to complete tasks, offering a quantitative measure of user interaction efficiency.

We chose the Keystroke-Level Model (KLM) over other evaluation methods like the NASA-TLX because our primary goal was to measure and compare the task efficiency of two different design methods when used by experts. KLM is ideal for predicting the time an expert user will take to accomplish a task without errors, as it breaks down the process into atomic physical and mental operators. In contrast, NASA-TLX measures subjective cognitive workload, which, while valuable for assessing user experience, is less suited for a direct, objective comparison of performance speed for a well-defined construction task. Given our focus on the efficiency gains of a novel authoring method, KLM provided a more direct and appropriate quantitative measure.

Evaluation Setup Participants were tasked with designing the same two interfaces featured in a previous ATB assessment [29]: a basic interface for a sequential problem and a more intricate interface for an arithmetic equation solver, both depicted in Figure 3.2. The time taken to complete each task was recorded to evaluate the efficiency of the two methods.

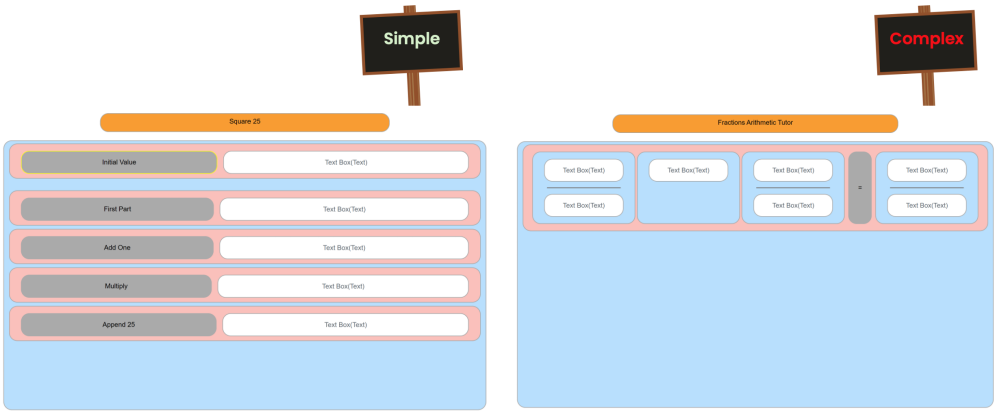


Fig. 3.2 This image depicts the two interfaces utilized in the assessment: On the left, the 'Simple' interface, created for sequential problem-solving tasks, provides a user-friendly design with basic input fields. On the right, the 'Complex' interface is customized for an arithmetic equation solver, showcasing a more sophisticated design with numerous input fields and operational functions for equation processing.

Results The use of AI assistance consistently decreased the time required to construct both simple and complex interfaces, as indicated in Table 3.1.

Interestingly, there was a more notable improvement in efficiency in the case of the complex interface, with a 68% decrease in time compared to the classical approach. This can be attributed to the complex interface, designed for an arithmetic equation solver, necessitating a more advanced layout featuring multiple input fields and operational functions. In this context, the AI assistance likely played a more significant role as users could utilize it to streamline the generation of equation components and layout elements more effectively.

Conversely, the efficiency enhancement for the simple interface was lower but still significant, at 23%. This lesser improvement can be elucidated by users having to manually input all the labels in the simple interface, a step that was not required for the complex interface. This manual input likely counteracted some of the efficiency benefits provided by the generative AI capabilities. Additionally, while the decrease in absolute keystrokes may not be as substantial as the time saved, it still contributes to the overall enhancements in efficiency.

Limitations Although this small-scale comparison offers promising insights into the potential of generative AI to enhance the efficiency of tutor interface design,

especially for complex tutors, further extensive studies involving diverse participants, tasks, and comprehensive feedback are essential to validate these findings and gain a deeper understanding of the tool's impact on user satisfaction and the design process.

3.3 Pedagogical Step Decomposition and Preference-Driven UI Refinement

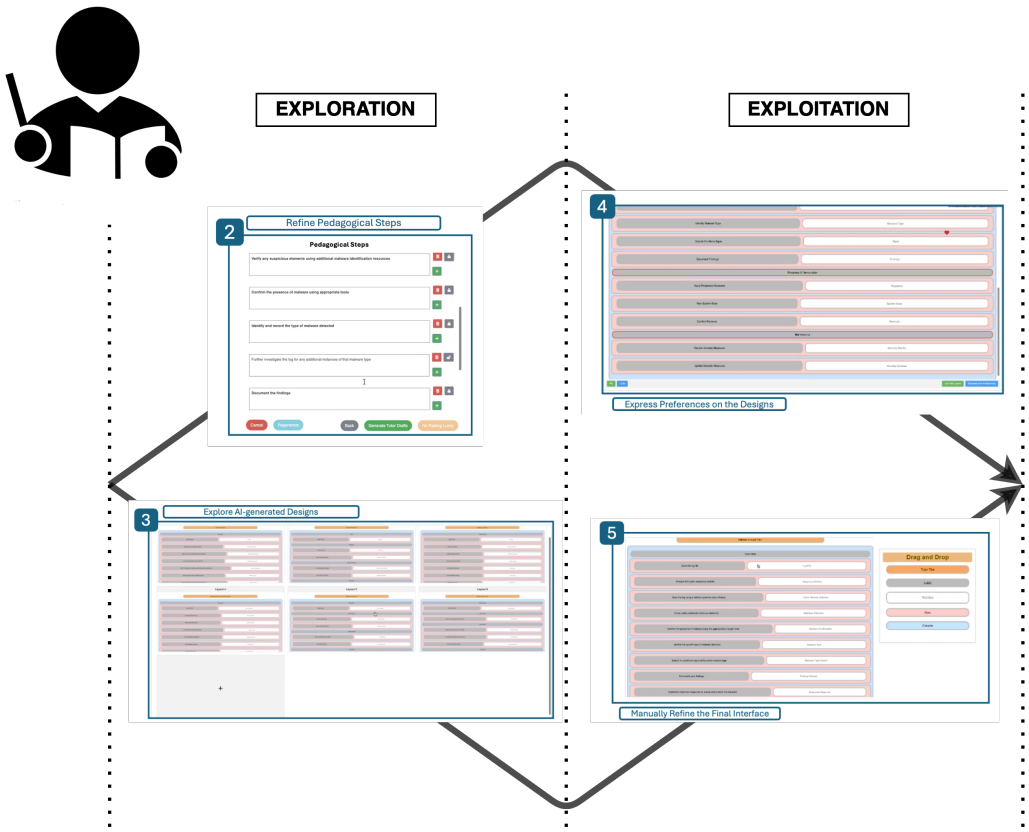


Fig. 3.3 Visual depiction illustrating how the AI-Assisted Tutor Builder facilitates the exploration-exploitation design cycle. The left side (Exploration) illustrates educators exploring possibilities through pedagogical step refinement (2) and multiple AI-generated designs (3). The right side (Exploitation) showcases educators exploiting preferred design elements through preference selection (4) and final refinement (5), establishing an efficient workflow for non-expert users.

Design Objectives

Our methodology is rooted in three fundamental design objectives: Pedagogical Alignment, Guided Flexibility, and Design Process Reflection. These objectives cater to the necessity of pedagogically appropriate interfaces, a harmonious blend of AI support, and accessible design procedures for educators lacking in-depth technical knowledge.

Pedagogical Alignment This objective ensures that both the design procedure and resultant interfaces adhere to educational principles and methodologies. It is implemented through the automatic breakdown of overarching tutoring objectives into organized, pedagogical phases, aligning with core ITS principles outlined by VanLehn [70]. This alignment serves a dual purpose: guaranteeing that the resulting tutoring interface inherently embodies effective pedagogical principles and offering a structured blueprint for interface development, where each decomposed phase informs specific UI components and their interrelations.

Guided Flexibility This objective harmonizes AI-driven automation with user empowerment and customization choices. The AI system formulates initial steps, allowing educators to precisely synchronize these steps with their pedagogical goals. The Guided Flexibility objective ensures that educators retain authority over key aspects of the tutoring process, while AI support can be utilized to explore alternate strategies and bolster overall coherence. This strategy aligns with Shneiderman's Human-Centered AI principles [82], ensuring educators maintain command over the process. By rendering the reflection of intricate design processes accessible, we strike a balance between AI-enabled automation and educator input, addressing the drawbacks of fully automated systems [77] that might overlook the specific requirements and preferences of novice users.

Design Process Reflection This objective ensures that users, regardless of their level of expertise in design, can partake in a structured design process that mirrors professional methodologies. By transforming complex design paradigms into an accessible, interactive framework, we enable educators to actively engage in the design process without the need for advanced skills. This objective is rooted in

the exploration-exploitation paradigm [102] and simplifies intricate design decision-making through AI-guided assistance, making it user-friendly for individuals without design backgrounds. It facilitates the exploration of various design possibilities and the refinement and exploitation of preferred design elements to customize the final interface to satisfy both pedagogical and aesthetic demands.

AI-Assisted Tutor Builder

In order to tackle the challenges in ITS development, we have introduced an AI-assisted tool that assists educators in a systematic manner to develop personalized tutoring systems. Our method follows a structured exploration-exploitation design paradigm (Figure 3.3) to lead educators through the process of creating interfaces. The left side of the workflow facilitates exploration by breaking down pedagogical aspects and offering various design options, while the right side supports exploitation by refining preferred elements with guidance. This method, which is both structured and flexible, makes intricate interface design procedures accessible to educators without a technical background.

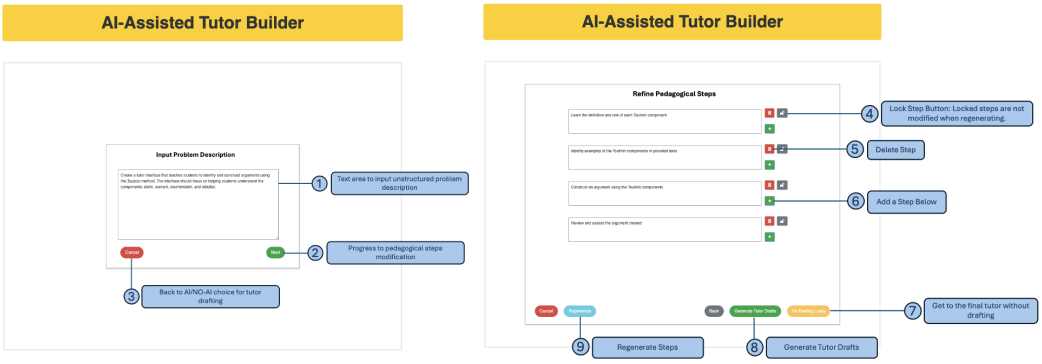


Fig. 3.4 (Left) Educators can enter unstructured problem descriptions in natural language through the requirements input interface (1), with options to advance to pedagogical steps (2) or return to previous stages (3). (Right) The pedagogical steps refinement interface provides various control options, such as locking critical steps (4), deleting unwanted steps (5), adding new steps (6), proceeding without drafting (7), generating tutor drafts (8), and regenerating steps while keeping locked content (9). This structured workflow helps educators move from high-level goals to concrete pedagogical steps.

Requirements Elicitation The initial phase of our system, as depicted in Figure 3.4, is the requirements elicitation section. This interface enables educators to input their high-level tutoring objectives and problem requirements using natural language. By offering an unrestricted input method, we empower educators to communicate their intentions without being limited to predefined formats or structures. This design decision is essential for ensuring the system's adaptability to a wide range of tutoring objectives in various subject areas, addressing a significant drawback of prior research that often concentrated on mathematical tutoring [77]. The flexibility of natural language input aligns with our aim to make the system user-friendly for educators from diverse fields without necessitating extensive technical expertise or domain-specific formatting.

Pedagogical Step Decomposition and Refinement After capturing the initial requirements, our system utilizes GPT-4 to break down these high-level objectives into specific, sequential pedagogical steps. This breakdown is illustrated in Figure 3.4. Each step is presented as an editable text field, allowing educators to adjust, remove, or insert steps as required. An important aspect of this interface is the capability to "lock" specific steps, denoted by a lock icon beside each step. By locking a step, educators indicate to the system that the step is crucial and should be retained in any subsequent iterations. This implementation directly supports our design objective of Guided Flexibility, striking a balance between AI-driven automation and educator control and customization choices.

The interface also contains buttons for "Regenerate" and "Generate Tutor Design." The "Regenerate" feature instructs GPT-4 to produce new steps for any unlocked items while keeping the locked steps unchanged. This enables educators to investigate different decompositions while retaining essential aspects of their pedagogical framework.

Interface Generation and Exploration When the "Generate Tutor Drafts" button is clicked, our system employs GPT-4 to generate various interface versions based on the refined pedagogical steps. These versions are displayed in a gallery format, as depicted in Figure 3.4. This gallery empowers educators to explore diverse design options without the need for extensive UI design knowledge. Each interface iteration

in the gallery is interactive, allowing educators to visualize how different layouts and components might operate in real-world scenarios.

Preference Selection and Refinement Figure 3.5 illustrates our innovative preference selection interface. This interface enables educators to indicate their preferences for specific elements across the generated designs. Educators can "pin" elements they want to keep in the final interface, denoted by a pushpin icon. These pinned elements are ensured to be included in the final design. Moreover, educators can "like" elements, indicated by a heart icon, to show a preference for their inclusion. The system strives to integrate liked elements into the final design, unless they conflict with the pinned elements. This preference-driven approach supports our Design Process Reflection objective by allowing educators to explore multiple design choices simultaneously.

Final Refinement with Traditional Tools After the stage of preference selection, the final interface design is produced by our system, taking into account the elements and preferences selected by the educator. As illustrated in Figure 3.5, educators can utilize conventional drag-and-drop tools to enhance this design further. This ultimate phase enables precise adjustments to the interface, guaranteeing that educators maintain complete authority over the final outcome.

Example Workflow For instance, an English teacher aiming to build a tutor for identifying metaphors could start with the requirement: "A tutor that teaches students to identify metaphors in sentences." The system would decompose this into pedagogical steps like: "1. Define what a metaphor is. 2. Show an example of a sentence with a metaphor. 3. Ask the student to identify the metaphor in a new sentence. 4. Provide feedback." The teacher could then lock the definition and feedback steps but regenerate alternatives for the examples. From the refined steps, the system would generate several draft interfaces in the gallery. The teacher might 'pin' the instructional text from one draft and 'like' the input field style from another, leading to a final, refined interface that combines their preferred elements.

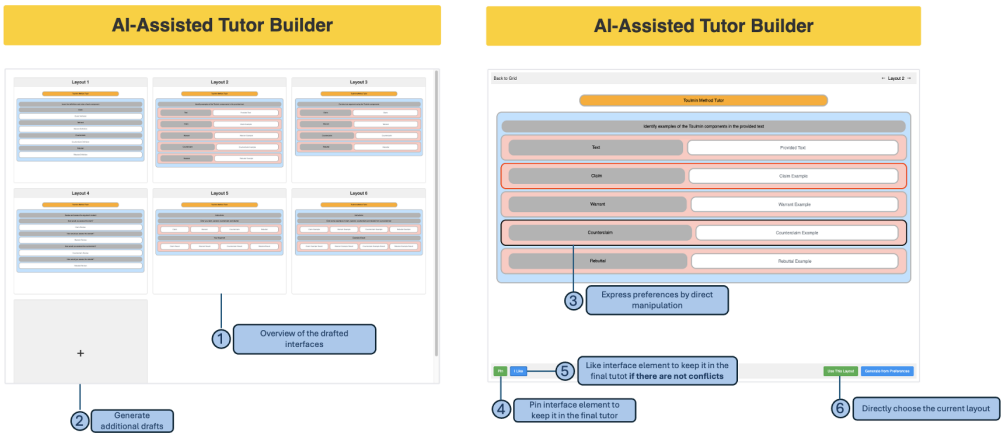


Fig. 3.5 (Left) The gallery of interface drafts shows multiple AI-generated layouts (1) with an option to create more variants (2). (Right) The interface for selecting preferences allows educators to directly adjust interface elements (3), with options to fix elements for guaranteed inclusion (4), prefer elements for potential inclusion if no conflicts exist (5), and make final layout choices (6). This two-pane view lets educators explore various design possibilities while retaining control over the ultimate interface composition.

Technical Implementation

Our AI-Assisted Tutor Builder was developed by combining JavaScript for the front-end and Python with Flask for the back-end. The foundation of our system is based on prompts tailored for GPT-4,[2] each created to enhance different functionalities of our system.

- **Tutor Steps Generation Prompt:** A prompt for creating a series of high-level steps for problem-solving. The emphasis is on clarity, conciseness, and adherence to pedagogical best practices.
- **Locked Steps Regeneration Prompt:** A prompt for generating new steps while retaining specific "locked" steps selected by the educator.
- **Layout Generation from Steps Prompt:** A prompt for producing multiple interface variations based on the refined pedagogical steps. These interfaces are generated using a Domain Specific Language (DSL) [77], a concise representation of the tutor interface layout. The prompt provides detailed instructions on the DSL format (e.g., utilizing row{}, column{}, label[], input[]) and design

[2]version gpt-4-0613

principles (e.g., segregating input elements, preserving equation integrity). The DSL is translated into HTML code to display the tutor interface.

- **Preference-Driven Layout Refinement Prompt:** A prompt for creating a final interface that combines elements from various tutors' layouts according to educator preferences.

Representative Prompt Examples To provide a clearer understanding of the interaction with the LLM, below are illustrative snippets of the prompts used in our system.

1. Tutor Steps Generation Prompt: System: You are an expert instructional designer. Your task is to take an educator's high-level goal for a tutor and break it down into clear, sequential, pedagogical steps. — User: Create a tutor to teach how to calculate the area of a rectangle. — Assistant: 1. Introduce the formula for the area of a rectangle (Area = length × width). 2. Provide a worked example with a rectangle of length 5 and width 3. 3. Ask the user to calculate the area for a new rectangle with length 8 and width 4. 4. Provide feedback on the user's answer.

2. Layout Generation from Steps Prompt (Simplified): System: You are a UI designer that generates tutor interfaces using a specific DSL. The DSL uses 'row' and 'column' for layout, 'label[]' for text, and 'input[]' for input fields. Based on the pedagogical steps provided, generate a DSL representation. — User: Steps: 1. Ask for the length. 2. Ask for the width. 3. Display the area. — Assistant: column row label[Enter length:] input[length] row label[Enter width:] input[width] row label[The area is:] label[result]

3.3.1 Methodology

In order to investigate the efficiency of AI-supported design tools in developing interfaces for Intelligent Tutoring Systems (ITS), a comparative user study with educators was carried out. The methodology employed a mix of quantitative metrics and qualitative observations to offer a thorough analysis of educators' interactions and the effectiveness comparison between conventional and AI-supported design approaches.

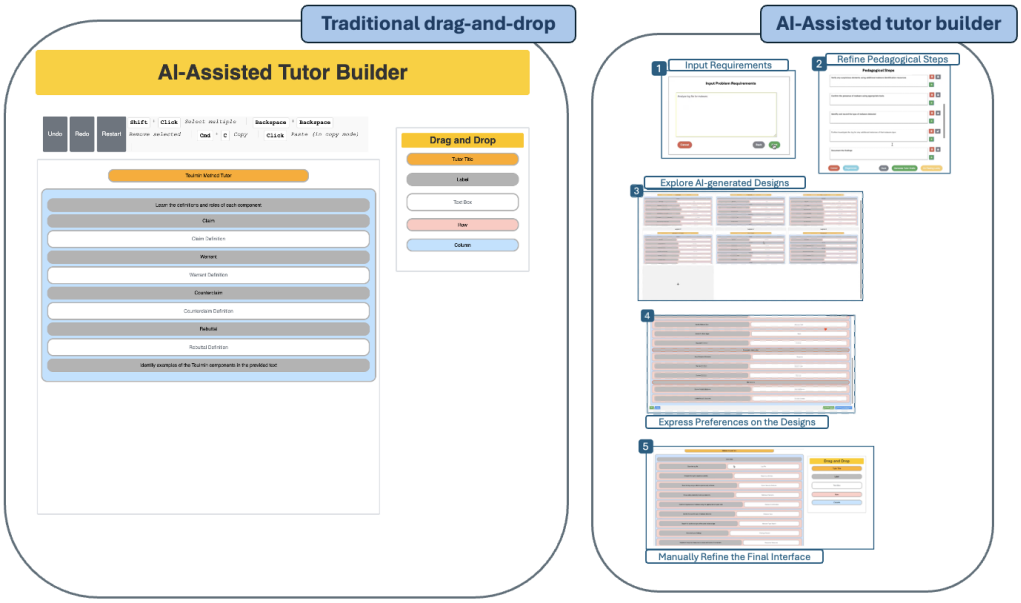


Fig. 3.6 Comparison of the interface design methods assessed in our research: (Left) Conventional drag-and-drop interface builder offering direct manipulation of interface elements and layout controls. (Right) AI-Assisted Tutor Builder demonstrating the complete workflow from requirements input (1) through pedagogical step refinement (2), exploration of AI-generated designs (3), preference expression (4), and final manual refinement (5). This side-by-side presentation highlights the main distinctions in approach between traditional and AI-assisted methods.

Participants

The participants were selected through a mix of snowball and convenience sampling, targeting K-12 educators who had an interest in or prior experience with educational technology. Those interested in participating filled out an initial questionnaire that included:

- *Demographic and background information:* age, gender, years of teaching experience, subject area, and experience with educational technology.
- *Technology proficiency:* self-rated proficiency with computers and educational software (novice, intermediate, advanced).
- *Interface design experience and AI familiarity:* previous experience with creating educational interfaces or materials and level of familiarity with AI technologies (novice, developing, competent, advanced, expert).

Table 3.2 A summary of the eight educators involved in our user study, detailing their demographic data, field of specialization, years of experience (YoE), and self-reported familiarity with AI tools and UI design.

ID (Gender, Age)	**Area of Expertise**	**YoE**	**AI tools Experience***	**UI Design Experience***
P1 (F, 35-44)	English	10	●○○○○	●●○○○
P2 (M, 35-44)	English	20	●●○○○	●●○○○
P3 (M, 55-64)	Mathematics	27	●○○○○	●○○○○
P4 (M, 45-54)	Cybersecurity	15	●●●○○	●○○○○
P5 (M, 35-44)	Computer Programming	6	●●○○○	●●●○○
P6 (M, 45-54)	Mathematics	14	●●○○○	●○○○○
P7 (F, 45-54)	Mathematics	15	●○○○○	●○○○○
P8 (M, 45-54)	Cybersecurity	25+	●●●○○	●●○○○

* ●○○○○ = novice ●●○○○ = developing ●●●○○ = competent ●●●●○ = advanced ●●●●● = expert

Criteria for inclusion in the study were as follows: a) current employment as teachers with a minimum of 1 year of experience; b) English proficiency; c) experience utilizing educational technology in teaching; and d) involvement in designing or creating digital instructional materials. A varied sample was sought, encompassing different subject areas, years of experience, and educational backgrounds to ensure diverse perspectives.

The final sample of eight K-12 educators (5 female, 3 male) was diverse in age (35-64 years), years of experience (M=16.5, range: 6-27), and subject area (English, Mathematics, Cybersecurity, Computer Programming). All participants had experience creating digital instructional materials, but none had prior experience

with ITS authoring tools. Their self-reported familiarity with AI tools and UI design varied, as detailed in Table 3.2.

The final sample comprised 8 participants (5 females and 3 males) aged between 35 and 64 years. The largest group (4) fell within the 45-54 age bracket, followed by 3 in the 35-44 range, and 1 in the 55-64 range. The sample reflected a mix of educational backgrounds, with 3 participants holding master's degrees, 3 holding doctoral degrees, and 2 possessing bachelor's degrees. Participants' teaching experience ranged from 6 to 27 years, averaging 16.5 years. The sample included educators from various subject areas, such as computer programming (1), English (2), mathematics (3), and cybersecurity (2). All participants reported familiarity with different educational technologies, including creating digital presentations (8), using learning management systems (7), designing online assessments (7), and employing AI tools for content creation or grading (7). Moreover, 7 participants had experience in producing instructional videos, and 6 had developed or modified digital learning materials. Concerning more advanced educational technology usage, 5 participants had crafted or personalized a course website, 4 had designed interactive digital activities (e.g., using H5P, Kahoot), and 3 had implemented adaptive learning technologies. Only 1 participant indicated using virtual or augmented reality tools in education.

In terms of user interface design experience, 4 out of 8 participants indicated having such experience. It is worth noting that none of the participants had prior experience with using or designing content for intelligent tutoring systems, marking this study as their initial encounter with this technology. The participants' familiarity with AI tools differed, with 2 participants having proficient experience, 3 having intermediate experience, and 3 having beginner-level experience. Table 3.2 offers a summary of the sample, detailing their fields of study, years of experience, and levels of proficiency in educational technology.

Method

The research was carried out using remote video conferencing software, divided into four primary stages:

Introduction and consent. Participants were informed about the study's objectives and processes, and gave their digital informed consent.

Interface design tasks. Each participant performed two interface design tasks:

- *Conventional approach:* Employing a drag-and-drop interface builder.
- *AI-driven approach:* Making use of our prototype AI-assisted design tool.

As depicted in Figure 3.6, these methodologies present fundamentally distinct approaches to interface creation - the conventional technique allowing direct manipulation of interface components, while the AI-assisted approach offers a structured workflow from requirements to final design. The sequence of methodologies was counterbalanced among participants to reduce the impact of learning effects. For each assignment, participants were given the freedom to outline their own requirements for an ITS interface relevant to their field of study. Subsequently, they were instructed to develop the most efficient interface within a 30-minute timeframe.

Post-task survey. Following each design task, participants filled out the System Usability Scale (SUS) [104], a standardized 10-item questionnaire with established benchmarks, to evaluate the overall usability of each approach.

Semi-structured interview. After both tasks, we carried out a 30-minute interview to collect detailed insights into participants' experiences, preferences, and suggestions for enhancement.

The design process was captured by screen-recording all sessions, and interviews were audio-recorded for subsequent analysis. Each participant's study session lasted around 90 minutes on average (M = 87 minutes, SD = 8 minutes).

Measurements

We gathered both quantitative and qualitative data to thoroughly evaluate the efficacy of each design approach:

Duration of task completion. The time taken to finish each interface design task was measured, starting from when the requirements were given until the participant indicated completion or when the 30-minute time limit was reached.

Quality of interface. Two expert raters in educational interface design independently assessed the quality of each completed interface using a standardized rubric that evaluated five key factors on a 5-point scale: coherence of layout, visual appeal, alignment with pedagogical principles, usability, and organization of content. Each factor was rated from 1 (poor) to 5 (excellent), with specific criteria for each level. The total score varied from 5 to 25, with ratings falling into categories of Excellent (21-25), Good (16-20), Adequate (11-15), Needs Improvement (6-10), or Poor (1-5). Inter-rater reliability was determined using weighted Cohen's kappa, applied to individual category scores to ensure evaluation consistency.

Usability ratings. The System Usability Scale (SUS) scores provided a quantitative measure of perceived usability for each design method.

Qualitative input. The semi-structured interviews investigated participants' experiences, preferences, perceived strengths and weaknesses of each method, and suggestions for enhancement.

Throughout the AI-assisted design task, we documented particular ***interactions*** with the AI tool, such as:

- Count of elements "liked" by participants in the generated interface drafts
- Count of elements "pinned" (selected to be kept) in the final interface design
- Frequency of deleting AI-generated steps
- Frequency of locking specific steps to prevent changes during regeneration
- Number of times participants chose to regenerate steps
- Frequency of editing AI-generated steps

The metrics offer understanding of participants' interaction and refinement of the AI-generated content, aiding in comprehending their design process and preferences while utilizing the AI-assisted tool.

3.3.2 Data Analysis

Our data was analyzed through a mixed-methods approach, combining quantitative statistical analyses with qualitative examination of participant feedback.

Quantitative analysis: The System Usability Scale (SUS) scores were analyzed using Wilcoxon signed-rank tests, with Benjamini-Hochberg correction applied at $\alpha = 0.10$ to handle multiple comparisons while maintaining appropriate power for our user study sample size. Paired t-tests were used to compare task completion times and interface quality measures between the traditional drag-and-drop method and the AI-assisted method. This analysis encompassed both overall interface quality scores and individual rubric items. Inter-rater agreement was determined using Weighted Cohen's Kappa, which is suitable for the ordinal data utilized in our interface evaluation rubric. Descriptive statistics were generated for the interaction metrics recorded during the AI-assisted design task, including the frequency of element likes, pins, step deletions, locks, regenerations, and edits.

Qualitative analysis: The focus of our analysis of the interview data was on extracting relevant quotes and insights that reflected participants' experiences with both design methods. We examined the interview transcripts to identify statements that offered detailed descriptions of participants' thoughts, preferences, and suggestions regarding the traditional and AI-assisted design processes. These quotes were then grouped thematically to enhance and provide context for understanding participants' interactions with and perceptions of the two design methods. We employed a thematic analysis approach. Two researchers independently reviewed the interview transcripts to identify initial codes and emergent themes. They then met to compare their findings, discuss discrepancies, and collaboratively develop a unified coding scheme. This consensus-based process, a standard practice for ensuring reliability in qualitative research, resulted in the final set of themes presented in our results.

3.3.3 Results

We present our findings across several dimensions, including user interactions, perceived usability, and overall effectiveness.

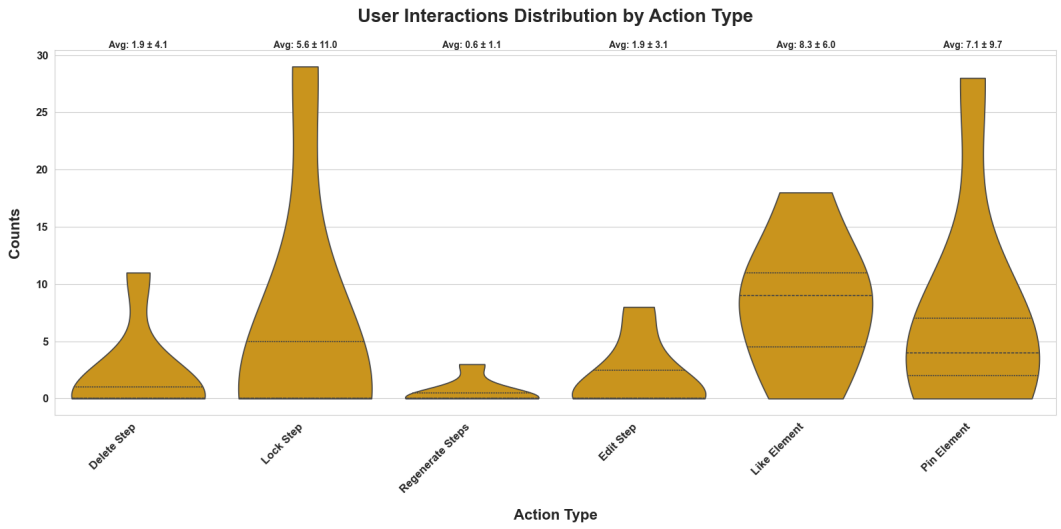


Fig. 3.7 Distribution of user interactions with AI-assisted design tools, including actions like 'Like Element', 'Pin Element', 'Delete Step', 'Lock Step', 'Regenerate Steps', and 'Edit Step' across eight participants. The average (Avg) and standard deviation (SD) for each action type are highlighted above the respective violin plots.

User Interaction Patterns

To gain insights into how educators engaged with the AI-assisted design tool, we recorded specific interactions during the design process, including the frequency of element likes, pins, step deletions, locks, regenerations, and edits. These metrics provide a window into participants' choices when using the AI-assisted tool, allowing us to understand the effectiveness of different features and identify areas for improvement. Figure 3.7 provides a detailed breakdown of how participants interacted with the AI-assisted design tool. In the step refinement phase, we observed varied usage of the features. The "Lock Step" feature was used moderately, with an average of (average=5.6, SD=11.0) locks per session. This wide standard deviation suggests significant variability in usage across participants. The "Edit Step" and "Delete Step" features were used more conservatively (average=1.9, SD=3.1) and (average=1.9, SD=4.1) per session, respectively. The "Regenerate Steps" feature was used least frequently (average=0.6, SD=1.1) times per session. In the interface design phase, the "Like Element" feature was heavily utilized across all participants, with an average of (average=8.3, SD=6.0) likes per session. The "Pin Element" feature, used to definitively select elements for the final design, saw slightly less frequent but still significant use (average=7.1, SD=9.7) pins per session. The higher usage of "Like"

compared to "Pin" suggests that while participants were comfortable expressing preferences through likes, they were somewhat more cautious about making definitive selections. The variability in feature usage, particularly for "Lock Step" and "Pin Element", highlights areas where user understanding and confidence in using certain features may vary. This suggests potential for improving feature explanations or user onboarding in future iterations. These interaction patterns demonstrate that participants actively engaged with the proposed features to realize our design goals. They utilized the AI-assisted tools in both the step refinement and interface design phases to shape their ITS interfaces according to their pedagogical needs and design preferences.

Perceived Usability and User Experience

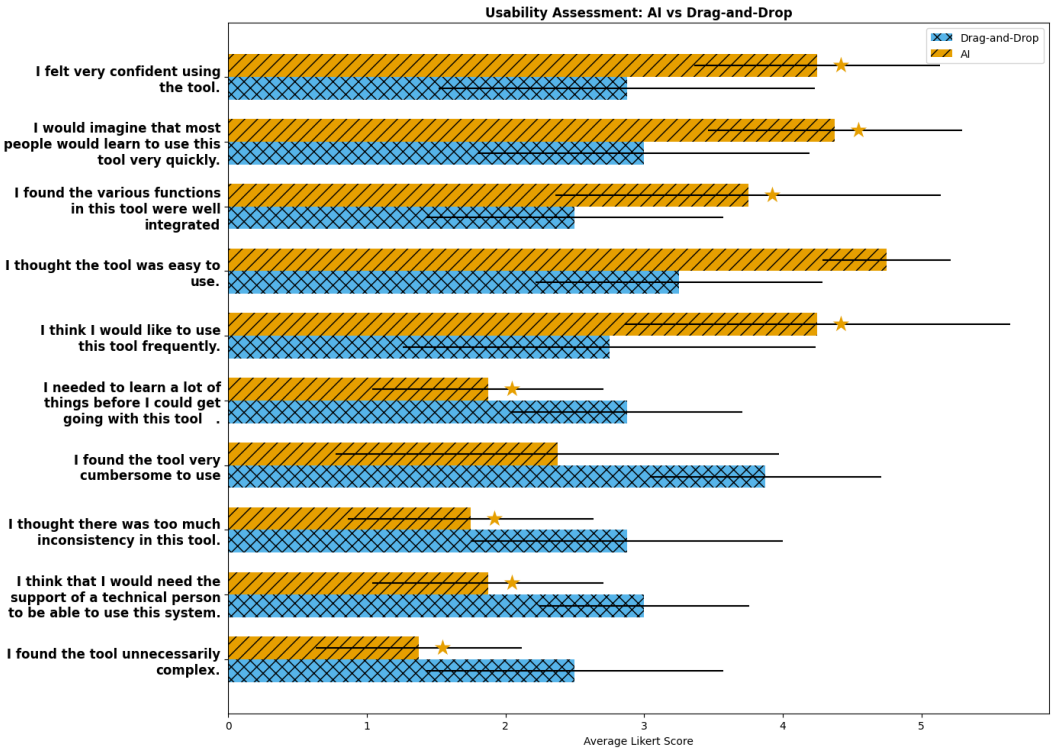


Fig. 3.8 Usability assessment comparing AI-assisted design with traditional drag-and-drop methods, highlighting aspects like confidence in use, learning curve, and perceived tool complexity. Error bars represent standard deviations. Asterisks (*) indicate statistically significant differences between the two methods ($p < 0.05$).

Table 3.3 Comparison of SUS items between AI-Assisted and No-AI (Drag-and-Drop) methods. Data are reported as mean (SD). Items marked with an asterisk (*) did not remain significant under Benjamini-Hochberg correction.

Item	AI-Assisted	No-AI	p-value	Effect Size
Complex	1.38 (0.74)	2.50 (1.07)	0.041	-0.625
Need Support	1.88 (0.83)	3.00 (0.76)	0.034	-0.625
Inconsistency	1.75 (0.89)	2.88 (1.13)	0.038	-0.625
Initial Learning	1.88 (0.83)	2.88 (0.83)	0.038	-0.625
Use Frequently	4.25 (1.39)	2.75 (1.49)	0.042	0.625
Better Integrated	3.75 (1.39)	2.50 (1.07)	0.039	0.625
More Confident	4.25 (0.89)	2.88 (1.36)	0.026	0.750
Faster Learnability	4.38 (0.92)	3.00 (1.20)	0.042	0.625
Ease of Use*	4.75 (0.46)	3.25 (1.04)	0.016	0.875
Cumbersome*	2.38 (1.60)	3.88 (0.83)	0.024	-0.750

$*$ Not significant after Benjamini-Hochberg correction ($\alpha = 0.10$).

To assess the usability and user experience of the AI-assisted method compared to the traditional drag-and-drop approach, we employed the System Usability Scale (SUS) [104]. Table 3.3 presents the results of the assessment. Overall, participants rated the AI-assisted tool more favorably than the traditional drag-and-drop method across most SUS dimensions. Educators reported that the AI-assisted approach was less complex, required less technical support, and was more consistently integrated into their workflows. They also indicated greater confidence and a stronger preference for frequent use under the AI-assisted condition, suggesting that it better aligns with educator-centered design needs. Although items related to ease of use and perceived cumbersome use reached nominal statistical significance before multiple-comparison adjustment, they did not remain significant under the Benjamini-Hochberg correction. Nonetheless, these results underscore the AI tool's potential for reducing complexity, enhancing educator confidence, and streamlining the ITS interface design process.

Interface Quality and Design Efficiency

Figure 3.9 presents the comparison of interface quality ratings and time taken to complete the design tasks.

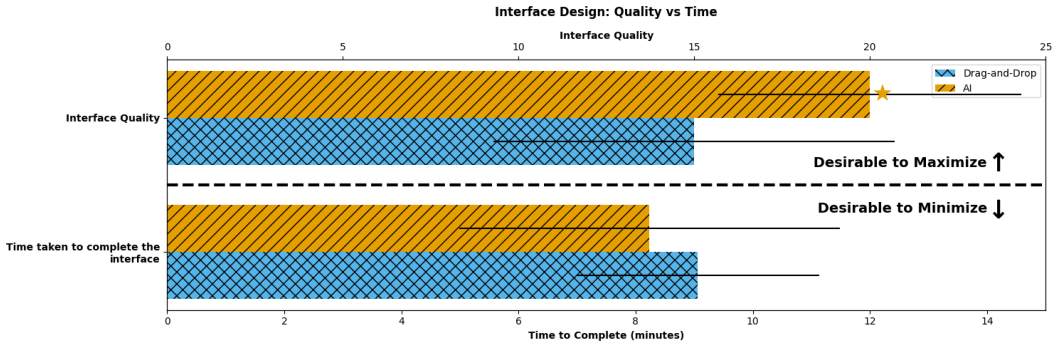


Fig. 3.9 Comparison of interface design quality and time efficiency between AI-assisted and drag-and-drop methods. Error bars represent standard deviations. Asterisks (*) indicate statistically significant differences between the two methods ($p < 0.05$).

Table 3.4 Comparison of interface quality criteria between AI-assisted and Drag-and-Drop methods. Data are reported as mean (SD).

Criterion	**AI-Assisted**	**No-AI**	**t(14)**	**p-value**
Layout Coherence	4.36 (0.72)	3.21 (1.15)	3.12	<0.01
Alignment with Pedagogical Principles	4.00 (1.07)	2.93 (1.39)	2.24	<0.05
Usability	3.86 (1.12)	2.86 (1.25)	2.18	<0.05
Content Organization	4.00 (1.25)	2.86 (1.46)	2.17	<0.05
Visual Appeal	3.79 (0.77)	3.14 (1.25)	1.63	>0.05

The AI-assisted method maintained comparable design time (AI average: 8.24 minutes, SD = 3.25; Drag-and-Drop average: 9.06 minutes, SD = 2.07; $t(14) = -0.60$, $p > 0.05$) while significantly enhancing interface quality (AI average: 20.00, SD = 4.31; Drag-and-Drop average: 15.00, SD = 5.71; $t(26) = 2.61$, $p < 0.02$). This combination of quality improvement and the superior usability scores demonstrates the benefits of the AI-assisted approach.

Figure 3.10 presents a comprehensive analysis of interface quality across five key criteria. In addition, Table 3.4 summarizes the quantitative comparison between the AI-assisted and traditional drag-and-drop methods. The AI-assisted approach demonstrated significantly higher ratings in Layout Coherence, Alignment with Pedagogical Principles, Usability, and Content Organization, indicating that AI-generated interfaces tend to be better aligned with educational principles, and more user-friendly. Although Visual Appeal scores were also higher for AI-generated

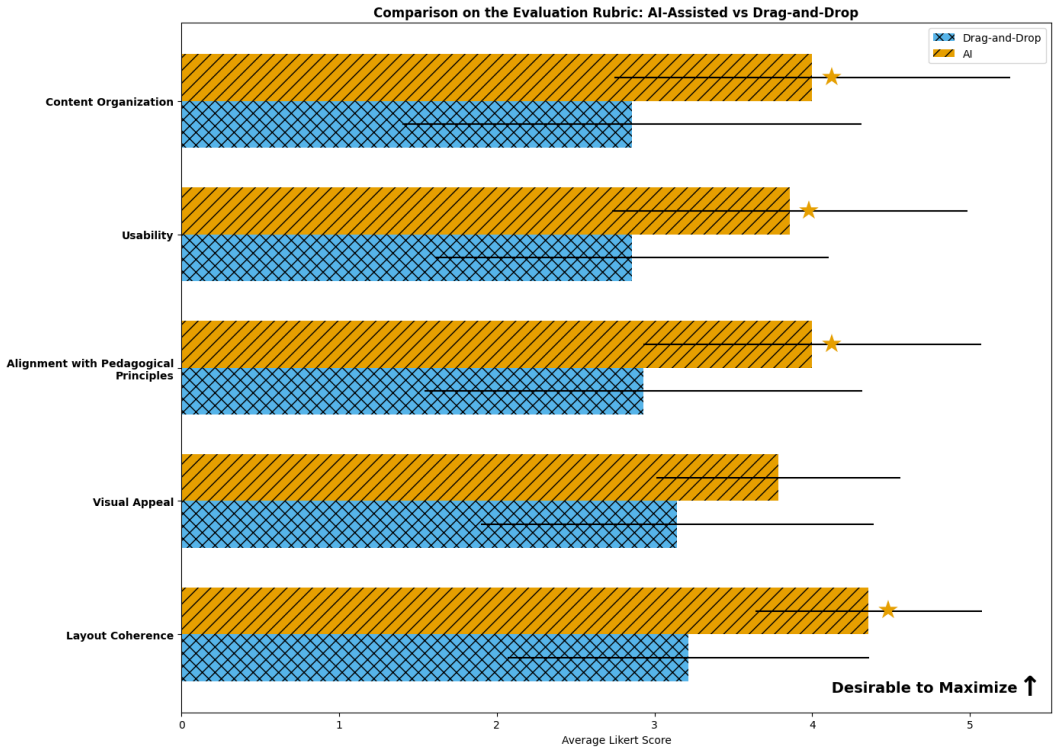


Fig. 3.10 Comparison of AI-Assisted vs Drag-and-Drop methods on the Evaluation Rubric. Error bars represent standard deviations. Asterisks (*) indicate statistically significant differences between the two methods ($p < 0.05$).

interfaces, this difference did not reach statistical significance. These findings suggest that while the AI-assisted method notably improves the functional aspects of interface design, its impact on aesthetics is less pronounced. Inter-rater reliability was assessed using the Intraclass Correlation Coefficient (ICC2) to account for the continuous nature of our evaluation criteria. The results showed varying levels of agreement between reviewers across different criteria: Layout Coherence: ICC2 = 0.26 (fair agreement), Visual Appeal: ICC2 = 0.06 (slight agreement), Alignment with Pedagogical Principles: ICC2 = 0.25 (fair agreement), Usability: ICC2 = 0.24 (fair agreement), and Content Organization: ICC2 = 0.35 (fair agreement). Despite these variations in interrater reliability, the consistent trend of higher scores for the AI-assisted method across all criteria strengthens the overall findings of the study.

Semi-structured Interviews

The semi-structured interviews offered valuable qualitative insights into participants' experiences with both the AI-assisted and the traditional drag-and-drop methods. These discussions captured the nuances of user interaction, gathered feedback on specific features, and highlighted perceptions of how an AI-assisted tool might impact the ITS design workflow.

A key theme was the positive user experience and enhanced usability of the AI-assisted approach. Participants found the new method more intuitive and efficient than drag-and-drop: *"AI-assisted method felt more intuitive and easier to use."* (P1). Many also emphasized time savings: *"It took away my inability to understand the drag-and-drop and replaced it with multiple layouts quite quickly."* (P4). Another educator (P5, Computer Programming) added, *"The AI felt like a collaborator. It did the grunt work of layout, and I could focus on the teaching flow, which is the part I actually care about."* The overall impression was that the AI-assisted tool allowed participants to focus more on educational goals instead of software details.

Another recurring theme involved the balance between user control and AI automation. While many felt they retained adequate control—*I was able to collaborate with the AI-assisted method to tailor the tutor to students' needs."* (P2)—others noted a reduction in manual oversight: *I had less control over the AI-generated final design compared to the drag-and-drop approach."* (P7). A math teacher (P6) elaborated, *"I liked the suggestions, but sometimes I just wanted to move a box one pixel to the left, and that felt harder with the AI. The drag-and-drop is clunky, but it's direct."* These differing perspectives imply that although automation streamlines the process, offering additional options for manual refinement might better meet diverse user preferences.

In terms of pedagogical impact, most participants agreed that the AI-assisted approach could lead to more effective and accessible tutor interfaces by lowering barriers for educators with minimal design expertise: *Instructors can generate content quickly without a lot of prior knowledge."* (P2). Yet, concerns were raised regarding instructional quality: *I did not like the limited explanations and instructions provided by the AI-assisted method."* (P3). This feedback indicates that while the tool enhances efficiency, further development is needed to ensure it fully supports sound pedagogical practices.

Looking forward, all participants expressed willingness to use an AI-assisted tool for tutor creation across diverse subject areas. They suggested enhancements such as more detailed instructions and improved visual design to further elevate the user experience. One participant remarked, *"More directions are needed to make full use of the tool."* (P5), emphasizing that the tool could be further refined fully support educators' needs. Another participant (P8, Cybersecurity) noted, *"This could be a game-changer for creating quick practice modules for my students before an exam. The potential is huge."* Overall, these implications underscore the importance of balancing automation with user control and comprehensive support to optimize both usability and pedagogical effectiveness.

3.3.4 Discussion

Our investigation into AI-supported interface design for Intelligent Tutoring Systems (ITSs) uncovers several key revelations regarding the transformative potential of AI in the realm of educational technology.

Enhanced Efficiency and User-Friendliness: Our AI-supported approach significantly enhances usability, as evaluated through both standard and bespoke metrics [104], in comparison to the prevailing drag-and-drop technique. The rapid generation of interface variations by AI empowered educators to explore a broad spectrum of design possibilities without requiring additional time commitments. This automation reduced manual labor, allowing educators to concentrate on higher-level pedagogical choices rather than the technical intricacies of interface design. The system's automated pedagogical step segmentation simplified the process by harmonizing interface elements with established instructional strategies, ultimately facilitating a more streamlined design process.

Improved Accessibility and Democratization: The notable improvement in user-friendliness is particularly significant, as the intricacy of ITS development had previously hindered its widespread adoption [26]. Our results suggest that AI support effectively mitigated this obstacle, rendering ITS creation more attainable for educators lacking specialized programming or design skills. This democratization of ITS development is in line with the objectives of prior authoring tools like CTAT [28] and ASPIRE [73], while pushing the field forward by directly addressing the specific challenges related to interface design. The capacity for educators to craft

tailored, context-specific tutoring materials holds promise for significantly enhancing educational practices by enabling more individualized, responsive, and culturally appropriate learning experiences in various educational environments.

Alignment of Quality and Pedagogy: The AI-assisted technique led to the development of interfaces that were more closely aligned with pedagogical objectives, a enhancement largely credited to our system's incorporation of scaffolding principles into the interface design process. While Calò and MacLellan [77] offered speed and direct translations of educator requests, our method introduced an additional level of pedagogical enhancement, guaranteeing that the interfaces were customized for specific educational settings. Moreover, participants frequently mentioned that the AI-generated interfaces encouraged them to explore design elements and teaching strategies that they may not have considered independently. This indicated that AI support could function not only as a tool for efficient execution but also as a wellspring of inspiration and growth for educators.

Challenges and Limitations: Although the majority of our findings were positive, it was crucial to recognize the challenges and limitations revealed in our research. The learning curve linked with the AI-assisted approach emphasized the importance of thorough onboarding and clear guidance during the introduction of innovative interaction models. This was consistent with broader challenges in incorporating new technologies into educational settings [27] and emphasized the vital role of user-centered design in AI-supported tools. A notable limitation was the lack of real-world testing for the final adoption of the developed tutor. Longitudinal studies would be necessary to evaluate the effectiveness and long-term engagement potential of the AI-generated interfaces. Our study was conducted with eight educators, which limited its generalizability. Additionally, the small sample size of eight participants may have hindered our ability to address educational areas where problems were challenging to resolve sequentially. In such instances, alternative interaction solutions would be required to cater to the distinctive pedagogical requirements and limitations of those fields. Subsequent research should encompass a more diverse range of demographics (gender, subject areas) and longitudinal studies to assess enduring usability.

Future Directions: Our study presents several promising avenues for future research and development in AI-assisted educational technology design. One primary direction is the investigation of more advanced preference learning algorithms that could further customize the AI's output to individual educators' styles and needs over

time. Research could explore how interaction with AI-generated designs impacts educators' own design thinking and pedagogical strategies over time. This could lead to the creation of AI-assisted tools that not only assist in developing educational resources but also actively support educators' professional growth. This should encompass not only the efficiency and quality of the design process but also the effectiveness of the resulting ITSs in actual classroom settings. Such research will be crucial in understanding the long-term implications of AI support in educational technology development and ensuring that these tools make a substantial contribution to enhanced learning outcomes.

Ethical Considerations

Our AI-assisted ITS design system enables educators to develop and customize tutor content, reducing dependence on pre-made, rigid materials. This strategy involves educators in the process, potentially reducing biases and other concerns that could arise from AI-generated content. By granting educators greater influence over tutor technology, we allow them to tailor the content to their specific teaching objectives and cultural contexts. As we progress towards wider implementation, we acknowledge the importance of ongoing ethical considerations. Future improvements may involve implementing bias detection algorithms to identify potential issues in design elements or content, such as stereotypical portrayals or unbalanced examples. Enhancing transparency in the AI decision-making process and expanding customization options will further empower educators to establish inclusive and equitable learning environments. Establishing ethical principles and training programs for educators regarding the responsible utilization of AI in education will be essential. Regular evaluations and feedback mechanisms involving various stakeholders will help ensure the system remains efficient and in accordance with evolving ethical standards in AI and education.

3.3.5 Summary

This chapter introduces the primary contribution aimed at addressing the difficulties related to accessibility and quality in ITS interface design with the assistance of AI. The objectives of our system design were threefold. Firstly, our aim was to aid educators in translating high-level instructional goals into practical, pedagogically

aligned interfaces. Secondly, we strived to offer adaptability and personalization, enabling educators to maintain authority over the design process. Thirdly, we concentrated on ensuring that the design process was intuitive and accessible, enabling educators with limited technical knowledge to actively engage in the development and customization of the interfaces. These goals were achieved by combining a user-centered approach with AI technology. Our system emphasizes a methodical pedagogical breakdown, which simplifies high-level instructional objectives into manageable, structured steps. Additionally, we implemented a UI refinement process based on user preferences, allowing educators to influence the AI's design decisions through user-friendly, click-based interactions.

In a user study comparing eight educators with different teaching backgrounds, our method maintained the same design time as existing drag-and-drop techniques but improved interface quality by 33%. The AI-assisted approach also showed improved usability, as indicated by higher scores on various aspects of the System Usability Scale. The main benefit of AI assistance is its capability to translate high-level instructions into lower-level elements (such as pedagogical steps and interface DSL). When coupled with human-centered AI principles, these tools have the potential to enhance end-user development by assisting non-expert users in overcoming technical challenges in creating digital content. This shift from abstract objectives to tangible outcomes, though requiring careful consideration of safeguards and debugging processes, could lead to more user-friendly, accessible, and efficient methods for generating educational materials. Ultimately, this empowers educators to play a more active role in customizing their teaching experience.

3.4 Conclusion

This chapter has presented novel approaches for empowering educators to create effective intelligent tutoring systems through AI-assisted interface design tools. Our contributions address a significant gap in existing ITS authoring platforms, which have historically focused on developing domain models and pedagogical strategies while assuming educators possess interface design expertise.

Our initial enhancement to the Apprentice Tutor Builder demonstrated significant efficiency improvements, reducing development time by 68% for complex interfaces while maintaining pedagogical quality. Building on this foundation, we introduced a

more comprehensive approach that combines pedagogical step decomposition with preference-driven UI refinement, allowing educators to maintain control over the design process while benefiting from AI-generated suggestions.

The user study involving eight K-12 educators validated our approach, showing a 33% improvement in interface quality compared to traditional drag-and-drop methods without increasing design time. The AI-assisted approach received significantly higher usability ratings across multiple dimensions, indicating that it successfully lowered technical barriers for educators without compromising their pedagogical agency.

These results suggest that AI-assisted authoring tools can democratize the creation of intelligent tutoring systems by making the development process more accessible to educators without technical backgrounds. By automating complex design decisions while keeping educators involved in the pedagogical structuring, our approach enables the creation of more personalized, context-appropriate learning experiences that reflect educators' teaching philosophies and domain expertise.

Future work will focus on expanding the system's capabilities to support a wider range of subject areas, developing more sophisticated preference learning algorithms, and conducting longitudinal studies to assess the effectiveness of educator-created tutoring systems in classroom settings.

Chapter 4

Web Development for End Users: Natural Language Tools for Creating and Customizing Websites

Publication notice. The techniques described in this chapter are from the work ("Leveraging Large Language Models for End-User Website Generation") published in *IS-EUD'23* and "MorphGUI: Real-time Natural Language Interface Customization with Large Language Models," currently under review.

4.1 Introduction

The widespread adoption of the Internet has fundamentally transformed how we live, work, and communicate, generating substantial demand for website development and customization. While web presence has become essential for individuals and organizations alike, traditional website development and customization remain technical endeavors that present significant barriers for non-technical users. This chapter explores how Generative AI can bridge this gap, enabling end-users to create and customize web interfaces using natural language without requiring programming expertise.

Traditional web development typically demands knowledge of HTML, CSS, JavaScript, and various frameworks—skills that are inaccessible to most people.

In response to this challenge, the field of End-User Development (EUD) [14, 105, 15, 106] has emerged, aiming to empower non-technical users to build websites without programming. While low-code/no-code tools [104, 107–110] have made progress by allowing users to visually arrange pre-built components with automatic code generation, these approaches present their own limitations. Many such tools impose steep learning curves, requiring significant time investment before users can create satisfactory results [16]. Additionally, they often constrain users to predefined options and templates, limiting flexibility for complex websites [111, 112, 17, 113].

Similarly, web interface customization remains challenging for end-users. Conventional one-size-fits-all graphical user interfaces (GUIs) fail to address diverse individual needs and context-dependent preferences [22, 21], resulting in suboptimal user experiences. Studies of interface personalization behavior reveal that users often want to modify interfaces in ways not anticipated by designers [18, 114]—such as reorganizing interface elements to match their workflow [20] or adjusting visual properties for accessibility [115]. Existing customization approaches typically require technical expertise with CSS and JavaScript, or they restrict users to predefined configuration options that fail to capture genuine adaptation requirements [19].

The emergence of Large Language Models (LLMs) has created unprecedented opportunities to address these challenges. LLMs trained on massive datasets can generate text that closely resembles human language, making them well-suited for translating natural language input into code [116–118]. Recent research has begun exploring LLMs for website generation [119] and code snippet creation [118], as well as interface customization through tools like Stylette [23] that enable styling adjustments through natural language. However, these early approaches have significant limitations—they typically do not allow users to refine generated output through iterative interaction, struggle with ambiguous specifications, or restrict customization to predefined attribute sets without supporting functional modifications or new component creation.

In this chapter, we present two complementary approaches that leverage LLMs to democratize web development and customization through natural language interaction. The first focuses on website creation, introducing a user-centric approach to LLM-driven web development that enables iterative refinement and multi-page support. The second addresses interface customization, presenting MorphGUI—a

framework that facilitates real-time GUI modification through structured natural language input.

Our LLM-driven web development approach centers around prompt engineering that guides model responses to follow predefined templates, enabling direct parsing and controlled modification. This allows users to concentrate on refining their websites conceptually without worrying about underlying code. Key innovations include a modification strategy that retains context and updates only specific code sections rather than regenerating entire documents, and support for creating multiple linked pages with shared context. The approach requires minimal technical knowledge, eliminating the need to master programming syntax or development tools, while maintaining user control through iterative refinement.

The MorphGUI framework complements website creation by enabling customization of existing interfaces through natural language. Unlike previous approaches that rely solely on predefined options or user models, MorphGUI integrates traditional GUI controls with LLM-powered customization. It employs a structured input approach that disentangles stylistic ("how it should appear") from functional ("what it should do") modifications, guiding users through the customization process and allowing targeting of specific components or the overall interface. This structured system enables MorphGUI to dynamically generate new interface elements or modify existing ones, providing flexibility beyond traditional predefined settings.

Both approaches represent significant advances in balancing automation with user control. While existing AI tools that transform hand-drawn designs into websites [120, 121] offer valuable alternative pathways for non-technical users, our natural language approaches provide distinct advantages. They enable iterative refinement, broader customization capabilities, and accommodate complex designs that might be difficult to express visually. Moreover, they align with the principles of Human-Centered AI [82], ensuring users retain control and understanding while benefiting from AI capabilities.

Through experimental evaluations, we demonstrate the effectiveness of these approaches in lowering technical barriers while preserving user agency. Our user study of MorphGUI with 18 participants shows that users regardless of technical expertise can successfully accomplish customization tasks through natural language, achieving an average target similarity of 73

Together, these approaches demonstrate how Generative AI can transform web development and customization from technical activities requiring specialized knowledge into accessible creative processes driven by natural language. By enabling users to express their intentions in familiar terms and providing appropriate guidance and feedback, we bridge the gap between human creativity and technical implementation, expanding who can participate in creating and customizing web interfaces.

4.2 LLM-Driven End-User Web Development

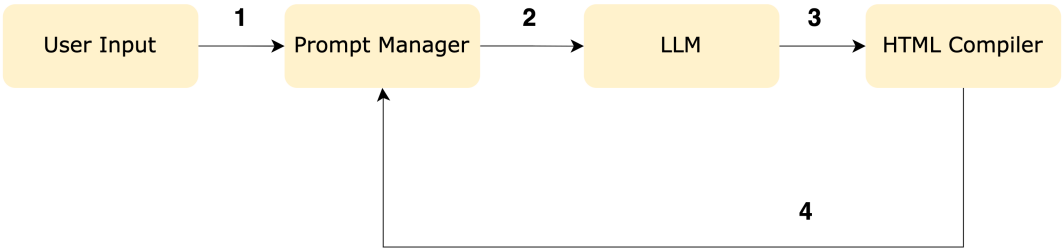


Fig. 4.1 The process of developing an interactive website: (1) User input, (2) Input processing by Prompt manager, (3) HTML code generation by LLM, (4) Error checking by HTML compiler, with a loop back to the Prompt manager for refinement if errors are found.

In our methodology, we employ a strategy that compels the LLM to adhere to a specific response template, as described in the provided prompt in Figure 4.2. The main objective is to guarantee that the produced code is organized and meets the criteria set by the user. The LLM is directed by a set of regulations that stipulate the structure of the generated responses. Through the utilization of this template-driven method, end users can concentrate on the desired functionality and design of their website without worrying about the underlying code, as the system processes the requests and responses in the designated format and converts them into HTML.

When initiating a new document, the LLM follows the response format: `new, ⟨document name⟩, ⟨code⟩`. For instances where a document necessitates alterations, the LLM complies with a response pattern that avoids displaying the entire code, opting for a more effective approach: `⟨document name⟩; ⟨add or replace⟩, ⟨n1-n2 range of lines if replace, n1 if add⟩,⟨code⟩; ... ⟨add or replace⟩, ⟨n1-n2 range of lines if replace, n1 if add⟩ ; ⟨code⟩`. This approach guarantees that only essential modifications are imple-

mented, preserving the original code and preventing unnecessary API responses. The LLM interprets the user's request as input, in the format `request:⟨request⟩`, and generates responses in accordance with the previously mentioned template. By adhering to the designated format, the LLM adeptly adjusts existing documents, only indicating the lines that were modified and the modifications required, while keeping the unchanged sections of the document intact.

Moreover, the method aids in error identification and resolution. The organized response layout enables the parser to pinpoint any mistakes in the produced code, equipping the user with the essential details to tackle these problems. When the LLM produces code with compilation errors, the system can address the issue directly by alerting the LLM about the error and the necessary code corrections, as shown in Figure 4.1. This efficient method for error detection and resolution conserves time and resources, enhancing the accessibility and efficiency of the web development process for non-technical users.

Prompt:
You have been tasked with generating HTML and CSS code according to the user's requirements. You can create multiple HTML files, but only one CSS file which will define the style of the page. I will provide you with the required response format, which you must strictly adhere to without including any additional words beyond the specified format.
If a new document needs to be created, your response should be in the format: new, ⟨document name⟩, ⟨code⟩.
If you need to make changes to a document that has been previously created in another response, you should not provide the entire code again. Even for substantial modifications, you should follow the format below for your response:
⟨document name⟩; ⟨add or replace⟩, ⟨n1-n2 range of lines if replacing, n1 if adding⟩, ⟨new line⟩; ... ⟨add or replace⟩, ⟨n1-n2 range of lines if replacing, n1 if adding⟩; ⟨new line⟩.
If no changes are needed for a specific document, you should not output anything.
When adding a line at a specified line number, indicate <add>, and when replacing a line within a specified range, specify <replace>.
Please note that the user's request will be provided as "request: ⟨request⟩".
Moreover, if you are modifying an existing document, only present the lines of modification and the modification itself in the prescribed format, as concisely as possible, to avoid re-entering unchanged parts of the document.

Fig. 4.2 The prompt designed to generate and edit HTML and CSS files according to natural language requirements, with specific response structures for making new files and revising current ones.

4.2.1 Method

Iterative Refinement and Multiple Pages Generation

The methodology enables users to refine the output of the LLM with subsequent input, giving them increased flexibility and control over the generated code. This iterative process ensures that the final website design closely aligns with the users'

requirements and preferences. Users can offer feedback and request changes in real-time, empowering them to actively influence the development process and avoid time-consuming revisions post website generation. Additionally, this approach facilitates the generation of multiple pages, enhancing the user experience and offering a more comprehensive solution for website development. Users can create interconnected pages with diverse designs and content, enabling the development of intricate and feature-rich websites without requiring extensive technical expertise.

Range of Design Choices

The proposed approach provides users with the ability to select from a range of design options in the generated documents. By offering various design alternatives, users can experiment with different aesthetics and layouts for their website, ensuring that the final outcome reflects their desired appearance. This functionality enhances the customization and flexibility of the website development process, enabling users to establish a distinctive and personalized online presence.

To assist in the choice of design options, the methodology may integrate predefined templates or design components that the LLM can utilize as a starting point. Users can then modify and personalize these templates according to their preferences, enabling them to swiftly create visually appealing websites without commencing from scratch. The LLM can also leverage user feedback during the iterative refinement phase to enhance the quality of the generated design options and cater to the specific requirements of the users.

Efficient Prompting Strategy

The strategy for efficient prompting involves instructing the LLM to provide only the count of lines to modify and the specific modifications, rather than rewriting the entire document. This method allows for generating code that benefits from a wider context window, as it involves fewer interactions and tokens. Consequently, the LLM can handle a longer sequence of refinements, which enhances model performance and user experience.

A fundamental issue in utilizing LLMs for website development is dealing with the constraints of the model's maximum token length. Other methods typically

entail generating the complete codebase at once, which may lead to surpassing the token limit. Through the implementation of the efficient prompting strategy, the approach enables the model to concentrate on the most pertinent sections of the code, thereby reducing the risk of exceeding the token limit. The reduction in the number of generated tokens is crucial for minimizing the expenses associated with API usage, as the costs are directly tied to the token count. This not only makes the approach more cost-effective for end-users but also facilitates broader utilization of LLM resources for website development.

Furthermore, a broader context window empowers the model to handle lengthier text sequences, enhancing its grasp of user requirements and its capability to produce precise and contextually appropriate code. Extending the context window also aids the LLM in maintaining consistency throughout the generated code, guaranteeing that the resulting website retains a uniform design and structure. With access to more contextual details, the LLM can make more informed decisions during code generation, thereby enhancing the overall quality of the produced website.

By creating a reduced number of tokens, the strategy put forward results in cost savings and improved resource utilization, as explained in the preceding sections. This not only enhances the affordability of the approach for end users but also enables broader utilization of LLM resources for website construction. The effective utilization of API resources may additionally result in quicker response times and a smoother user experience during interactions with the LLM.

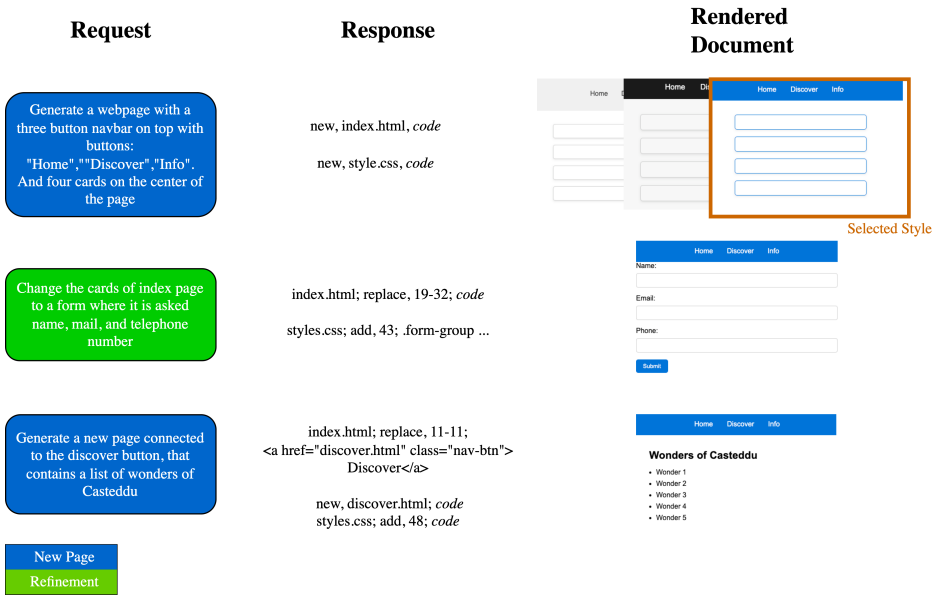


Fig. 4.3 Visualization of the sequential website development process, represented in columns and rows: Columns represent the Request, Response, and Rendered Page; Rows illustrate three requests – the first and third for creating new pages, and the second for improving the current page. It is important to note that the code in the response is not included, but the provided responses should help readers grasp the format used for interacting with the LLM.

4.2.2 Proof of Concept

In order to showcase the effectiveness of our method, we have developed a proof-of-concept implementation using GPT-4 [122], demonstrating the practical application of the technique. Figure 4.3 depicts the sequential instructions for website development using natural language and the GPT-4 model. The figure illustrates the interaction format, helping readers grasp the structured responses and the methodology employed. While the actual code within the responses is not displayed, the provided responses offer enough detail to understand the template and format utilized in the GPT-4 interaction. This proof-of-concept example demonstrates how our approach can be utilized in real-world scenarios to design and refine websites based on natural language specifications and the capabilities of LLMs, ultimately simplifying the web development process for users.

4.2.3 Summary

The suggested methodology presents an efficient prompting strategy for engaging with external LLM APIs, which aims to optimize resource utilization and enhance user experience. By concentrating on reducing the number of generated tokens and expanding the context window, this method facilitates cost savings, improved resource utilization, enhanced model performance, and greater control over the generated code.

Offering a range of design choices and iterative refinement further enhances the customization and adaptability of the website development process. By tackling key challenges in LLM-based website development, such as token restrictions and contextual comprehension, the proposed methodology is poised for integration with current low-code/no-code tools to extend the technology's benefits to a broader audience.

In future endeavors, integrating the approach with a low-code/no-code platform can validate the effectiveness and usefulness of the methodology through user studies. This integration can also streamline the development process by providing a user interface for interacting with the LLM, thereby enhancing the overall user experience. The ultimate objective remains to democratize website development and make it more accessible to users lacking technical expertise.

4.3 Real-Time GUI Customization with Natural Language

Traditional graphical user interfaces offer limited customization options, typically constraining users to predefined settings that fail to address their unique needs. While developers can modify interfaces through code, end-users without programming knowledge are often unable to implement the specific modifications they desire. Section introduces MorphGUI, a framework that bridges this gap by enabling real-time interface customization through natural language. By leveraging Large Language Models to interpret user requests and generate corresponding interface modifications, our approach empowers users to articulate and implement complex adaptations without technical expertise. The system's structured input method separates functional

requirements from visual specifications, guiding users through the customization process while maintaining interface coherence.

We present a use case scenario that follows a content manager named Sarah. This narrative demonstrates the limitations of conventional interface customization methods and highlights how our natural language customization framework addresses these constraints. By contrasting Sarah's experience with traditional GUI controls against her interaction with MorphGUI's structured natural language input system, we showcase how end-users can articulate and implement complex, context-specific interface modifications that go beyond predefined settings. This scenario emphasizes MorphGUI's ability to empower users without technical expertise to realize customization intentions that would otherwise require developer intervention.

Consider Sarah, a content manager who frequently utilizes a calendar application to schedule and track team meetings. While the application provides standard customization settings for colors, fonts, and basic layout options, Sarah faces difficulties in adapting the interface to meet her team's specific workflow requirements. She aims to alter how meetings are presented to emphasize upcoming deadlines and differentiate between various project categories.

Initially, Sarah tries using the conventional GUI controls to tweak color palettes and text sizes using preset configurations. However, she soon realizes that these fixed choices are insufficient for achieving her desired outcome: integrating deadline details with meeting titles and applying conditional formatting based on project types. Employing MorphGUI's structured input method, Sarah first opts for the "Global GUI" adjustment and defines in the "What it should do" field: "Display meeting deadlines alongside titles and group events by project category." In the "How it should appear" field, she specifies: "Utilize bold red text for urgent deadlines and employ distinct background colors for each project category." Through this natural language interaction, MorphGUI produces a fresh calendar view that merges deadline information with meeting presentations and enacts the requested visual classification. Sarah then selects the specific header component to introduce a new feature, articulating in the functional field: "Include a color legend displaying project categories" and in the visual field: "Present as a horizontal strip at the top featuring colored boxes and category labels." The structured input fields lead her through each adjustment while upholding the coherence of the overall interface.

This use case illustrates how MorphGUI goes beyond conventional static customization choices by enabling users to articulate and execute more intricate, context-specific interface adjustments using natural language directives, at both overall and component-specific levels.

4.3.1 Overview of the Architecture

Our system is designed with the architecture depicted in Figure 4.4. Users interact with the system through a web-based front-end that combines application-specific features with customization options for the interface.

The dynamic component module serves as the central visualization engine, overseeing both the rendering of the current interface and the creation of updated views based on user input. This module communicates directly with the front-end to provide real-time interface updates. It adopts a component-based architecture that permits the dynamic adjustment and re-rendering of interface elements without necessitating page reloads.

The server handles user preferences, database operations, historical system modifications, and coordination with the AI module. The AI component, which utilizes a Large Language Model, interprets natural language descriptions and transforms them into specific interface specifications. These specifications undergo validation before being converted into executable interface code by the dynamic component system.

A database layer manages the persistence of user preferences, interface versions, and component configurations. The system retains both current and previous interface states, enabling version control features that empower users to monitor and reverse changes as required. This versioning mechanism offers users a safety net when experimenting with interface alterations.

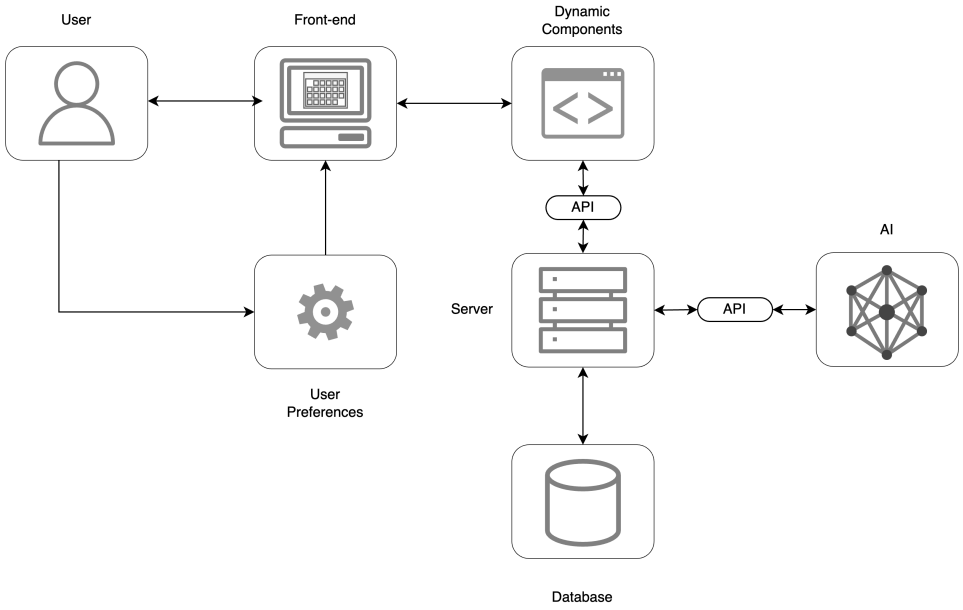


Fig. 4.4 Illustration of the system architecture depicting the connections among user interface, server elements, and AI services

Customization Interface of MorphGUI

The customization interface of MorphGUI, as depicted in Figure 4.5, is divided into two primary sections: traditional static settings and natural language-driven customization. The "Static Settings" dropdown allows users to utilize standard GUI controls for basic adjustments such as colors, fonts, and layout choices. The "Dynamic Settings" segment facilitates more intricate customizations through natural language input.

The approach of Large Language Models (LLMs) introduces new challenges in ensuring consistent interpretation of user intent and preserving interface usability throughout customizations [123]. Recent studies have demonstrated that prompt engineering is essential in translating abstract user objectives into tangible interface implementations, although creating effective prompts is not straightforward for many users [78, 124]. Fundamental principles for successful prompting with LLMs encompass offering clear guidance, utilizing relevant examples, dividing tasks into smaller components, and accurately detailing layout and styling requirements [125, 126, 119].

Our framework leverages these insights by incorporating a natural language interface that allows users to operate at various levels of customization. In Figure 4.5, within the "Personalize" section, users have the option to either modify existing interface elements, add new components, or select "Global GUI" to enhance the entire interface. Within the chosen scope, users can communicate their desired alterations through two distinct input fields that outline their functional and aesthetic criteria. The first field, titled "What it should do," addresses behavioral adjustments and functional requirements. The second field, "How it should appear," concentrates on visual aspects such as styling and layout. This approach of using two separate fields assists users in expressing their preferences effectively by offering clear contexts for functionality and appearance, allowing the system to handle behavioral changes and visual modifications independently. A button styled with a gradient initiates the customization process, and a "Previous" option permits users to revert changes if necessary.

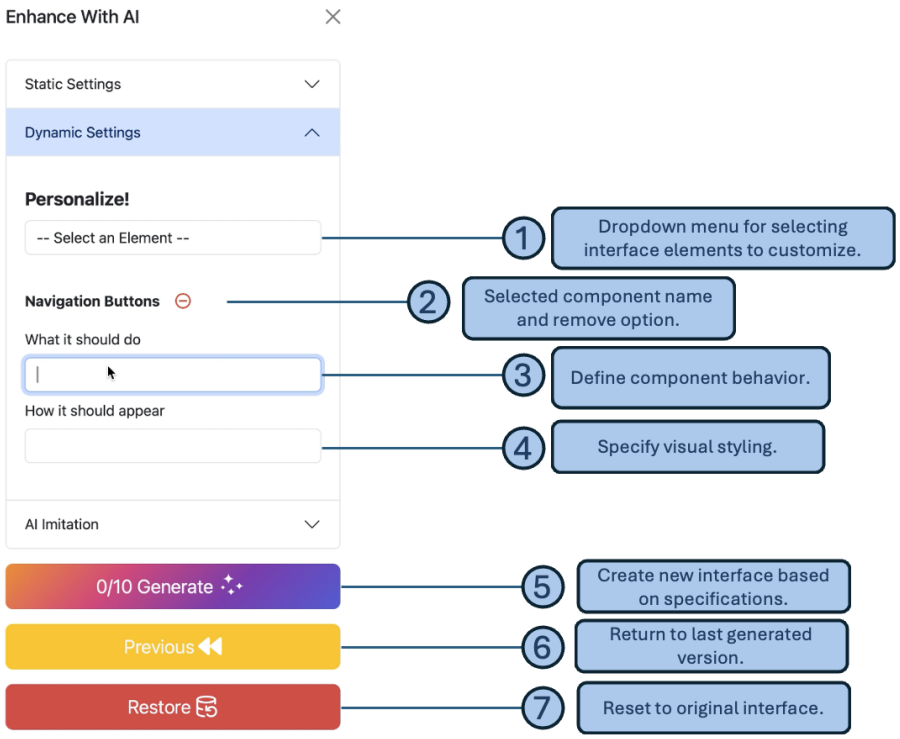


Fig. 4.5 Interface for customizing natural language, displaying options for selecting components and entering preferences

The engineering strategy prompt that forms the basis of this interface integrates the context of the selected component, user inputs, and technical limitations to create structured prompts for the Large Language Model. These prompts guarantee that any modifications generated uphold the functionality of the component while adhering to the specified changes within the constraints of the technical framework. For example, when Sarah provides her input, the system constructs a prompt for the LLM like this:

```
System: You are a React component generator. Modify the provided calendar
component code based on the user's request. The user wants to change the
'Global GUI'.

Component Context:
<current code of the calendar component...>

User Functional Request (What it should do):
"Display meeting deadlines alongside titles and group events by project category."

User Visual Request (How it should appear):
"Utilize bold red text for urgent deadlines and employ distinct background
colors for each project category."

Generate the new React component code.
```

4.3.2 Dynamic Component System

The system for dynamic components facilitates real-time updates for interfaces by overseeing the generation, assessment, and display of components while the program is running. At its core is a specialized React[1] component acting as a link between the produced interface code and the application's runtime environment. The management of runtime components is accomplished through a blend of code evaluation and dynamic rendering. Upon receiving new interface specifications, the system initially

[1] https://react.dev

preprocesses the code to ensure compatibility with the runtime environment. This preprocessing involves managing import statements, resolving dependencies, and ensuring seamless integration with the application's current component ecosystem. The component utilizes Babel[2] for code transformation and compilation, allowing it to securely evaluate and execute the produced code. Preserving the state is a crucial element of the system. While updating components, the dynamic component system upholds the application's state through React's lifecycle methods, ensuring that user data and interaction states endure through interface updates. The system incorporates a state tracking mechanism that conserves essential values during component regeneration. Specifically, component state is managed using React's 'useState' and 'useContext' hooks. When a component is about to be re-rendered with new generated code, its current state is serialized to a temporary in-memory store. After the new component is rendered, this state is restored, ensuring a seamless user experience without data loss during customization.

Code generation and evaluation take place through a pipeline that guarantees security and dependability. The generated code is validated prior to being converted into executable components. The system follows a structured method for creating components. Initially, code preprocessing manages dependencies and imports to ensure proper resource management. Subsequently, the code is transformed using Babel, which converts modern JavaScript features into compatible code. A new function component is then created through dynamic evaluation, enabling runtime component generation. Lastly, this newly generated component is integrated into the React component tree, facilitating rendering within the application's interface.

Styling and layout control are handled by both inline styles and dynamic CSS generation. The system accommodates various styling techniques, such as direct style objects and class-based styling, while maintaining consistency with the application's current style system. Style definitions are processed alongside component code to ensure accurate rendering.

This strategy for managing dynamic components empowers the system to deal with intricate interface updates while upholding application stability and performance. The division of responsibilities among component generation, state management, and style control permits adaptable and dependable interface customization.

[2] https://babeljs.io

4.3.3 Study Design

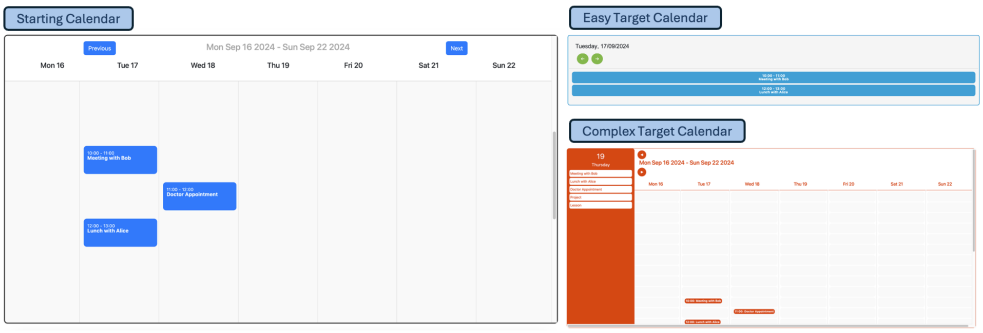


Fig. 4.6 Examples of calendar interfaces displaying the initial calendar (left) and the target calendars for simple (top right) and intricate (bottom right) customization assignments.

In order to assess MorphGUI, we carried out a within-subjects study with 18 participants who completed two customization tasks of increasing complexity. Figure 4.6 illustrates the initial calendar interface and the desired designs for both simple and intricate customization tasks. This approach enabled a direct comparison of participant performance and behavior across varying interface complexities, while accounting for differences in technical background and LLM experience.

We gathered both quantitative and qualitative data using various methods. Quantitative data included System Usability Scale (SUS) scores and task completion metrics. Qualitative feedback was obtained through open-ended questions in post-task interviews, focusing on participants' experiences with the customization process. Furthermore, we assessed the quality of customization outcomes by comparing the visual and functional similarities between participants' created interfaces and the target designs.

Participants

The study was carried out with 18 participants who were recruited using a mix of snowball and convenience sampling methods. Table 4.1 displays the breakdown of our sample in terms of age, professional background, and familiarity with Large Language Models. The age distribution revealed that 12 participants fell within the 20-30 age bracket, 5 were between 30-60 years old, and 1 was under 20. In terms of

Table 4.1 Summary of participants in the study (N=18), presenting demographic details, LLM usage, and levels of experience in UI development.

ID (Age)	Profession	LLM Usage Exp.*	UI Development Exp.**
P1 (20-30)	Technology	●●●○	●○○
P2 (20-30)	Technology	●●●●	●○○
P3 (20-30)	Technology	●●●●	●○○
P4 (20-30)	Design	●●●○	●●○
P5 (20-30)	Technology	●●●○	●○○
P6 (20-30)	Technology	●●●●	●●●
P7 (20-30)	Healthcare	●●●●	●○○
P8 (20-30)	Marketing	●●●○	●○○
P9 (20-30)	Healthcare	●●●○	●○○
P10 (20-30)	Technology	●●●●	●○○
P11 (<20)	Other	●●●●	●○○
P12 (30-60)	Other	●●○○	●○○
P13 (30-60)	Healthcare	●○○○	●○○
P14 (20-30)	Healthcare	●●●○	●○○
P15 (30-60)	Legal	●○○○	●○○
P16 (30-60)	Legal	●○○○	●○○
P17 (20-30)	Technology	●●●○	●○○
P18 (30-60)	Other	●○○○	●○○

* ●○○○= No Knowledge, ●●○○= Heard Of, ●●●○= Used Occasionally, ●●●●= Regular User

** ●○○= No Knowledge, ●●○= Basic Knowledge, ●●●= Proficient,

professional backgrounds, there was a range of diversity, with 9 participants coming from technology-related fields. The remaining participants represented different sectors such as legal (2), healthcare (2), design (1), marketing (1), and other fields (3), offering insights from various professional perspectives. Regarding experience with Large Language Models, 7 participants mentioned using them occasionally, while 6 were regular users. 4 participants had no prior exposure to LLMs, and 1 had only heard about them without direct interaction.

4.3.4 Method

The research was carried out remotely through video conference calls. At the beginning, participants were provided with an introduction to the study and were asked to fill out the necessary informed consent documents. The study was organized into three distinct stages:

Setup and Training. Prior to engaging with the system, participants were presented with an open-ended query concerning their preferences and requirements for customizing the calendar interface. This initial investigation sought to grasp the users' anticipations regarding interface personalization without influencing them with existing solutions, following established HCI methodologies for user-centered design. Subsequently, upon gathering their responses, participants were provided with a brief orientation to the interface customization system and a concise demonstration to acquaint themselves with the natural language input mechanism and generation process.

Customization Tasks. The participants undertook two tasks in sequence. Each participant commenced with the straightforward task and then progressed to the intricate task. Both tasks had a time limit of 20 minutes and allowed a maximum of 10 generation attempts. For each task, participants initiated with the same foundational calendar interface and endeavored to recreate a specified target design. Participants had the flexibility to revert to previous versions or restore the original interface at any point. The system recorded all interactions and generation attempts. The tasks were structured for sequential completion to enable participants to become accustomed to MorphGUI's natural language customization capabilities during the initial task. The consistent task order ensured that all participants received uniform learning opportunities, attributing the observed performance enhancements to increased familiarity with MorphGUI.

Evaluation and Interview. Upon finishing both tasks, participants completed the System Usability Scale (SUS) questionnaire [127] and engaged in a semi-structured interview regarding their encounter with the system. The interview centered on their customization approach, encountered challenges, and recommendations for enhancement.

Each session had a duration of around 45 minutes (M = 43 minutes, SD = 8 minutes). All sessions were recorded with the participants' consent for later analysis of interaction patterns and customization strategies.

Measurements

Throughout the study, we obtained three categories of measurements. Initially, system logs documented participant actions, such as session length, attempts made (out of 10 available per task), resets of the interface, reverting to previous versions, and system errors or warnings. These quantitative measurements offered insight into participants' interaction behaviors and system functionality.

Subsequently, upon completing both tasks, participants filled out the SUS questionnaire to assess the overall usability of the natural language customization system.

Lastly, participants responded to open-ended inquiries regarding their customization experience, concentrating on their utilization of natural language descriptions, encountered difficulties, and recommendations for enhancement. These replies provided qualitative perspectives on participants' cognitive models and interaction approaches.

4.3.5 Data Analysis

Analysis of system logs was conducted to determine the mean session duration, average number of generation attempts, and frequency of resets and reversions among participants. These metrics were compared between tasks using paired t-tests to assess differences in interaction patterns between simple and complex customization scenarios.

Scores for interface evaluation (on a 0-5 scale) were computed for each component and combined into a target adherence score. Mean scores and standard deviations were calculated for each component category and the overall adherence score. Pearson correlation coefficients were used to explore the relationships between participant characteristics (LLM experience, technical background) and customization success. Wilcoxon rank-sum tests were utilized for analyzing differences in performance across groups for categorical variables.

Responses to the SUS questionnaire were scored following standard procedures. Raw scores were normalized to a 0-100 scale, with mean scores and standard deviations computed. These scores were compared to established SUS benchmarks to assess system usability. Additionally, individual question scores were analyzed to pinpoint specific usability strengths and concerns.

Qualitative feedback from interviews and open-ended questions was used to present representative quotes and observations regarding participants' customization strategies and experiences with our system. Results were summarized using descriptive statistics, and statistical significance, when applicable, was reported at $p < .05$.

The final generated interfaces were evaluated against the target designs using a structured assessment framework that examined layout accuracy, component positioning, color scheme implementation, navigation functionality, and event display formatting. These scores were combined to form a target adherence score to assess participants' ability to achieve their customization objectives through MoldGUI.

4.3.6 Results

Task Performance

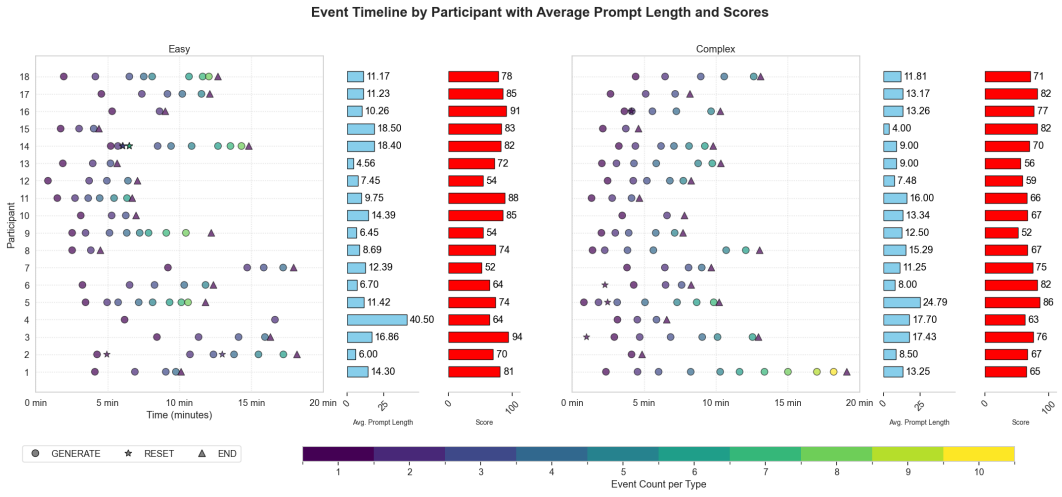


Fig. 4.7 Event Timeline by Participant with Average Prompt Length and Scores.

Participants successfully completed both customization tasks, with higher performance on the simple task (M=74.7%, SD=12.9%) compared to the complex task (M=70.2%, SD=9.6%). Task completion times decreased between the simple (M=16.7 min, SD=7.4) and complex tasks (M=13.6 min, SD=4.6), suggesting a learning effect. An overview of task performance and prompt behavior across participants is illustrated in Figure 4.7.

Interface Generation Patterns

Participants used an average of 5.0 (SD=2.0) generation attempts for the simple task and 6.0 (SD=1.9) for the complex task. Reset operations were rare (M=1.1, SD=0.8 per task), while reversions to previous versions were more common (M=2.1, SD=1.2). Analysis showed no significant correlation between number of generation attempts and final interface quality ($r = .21, p > .05$).

Natural Language Input Analysis

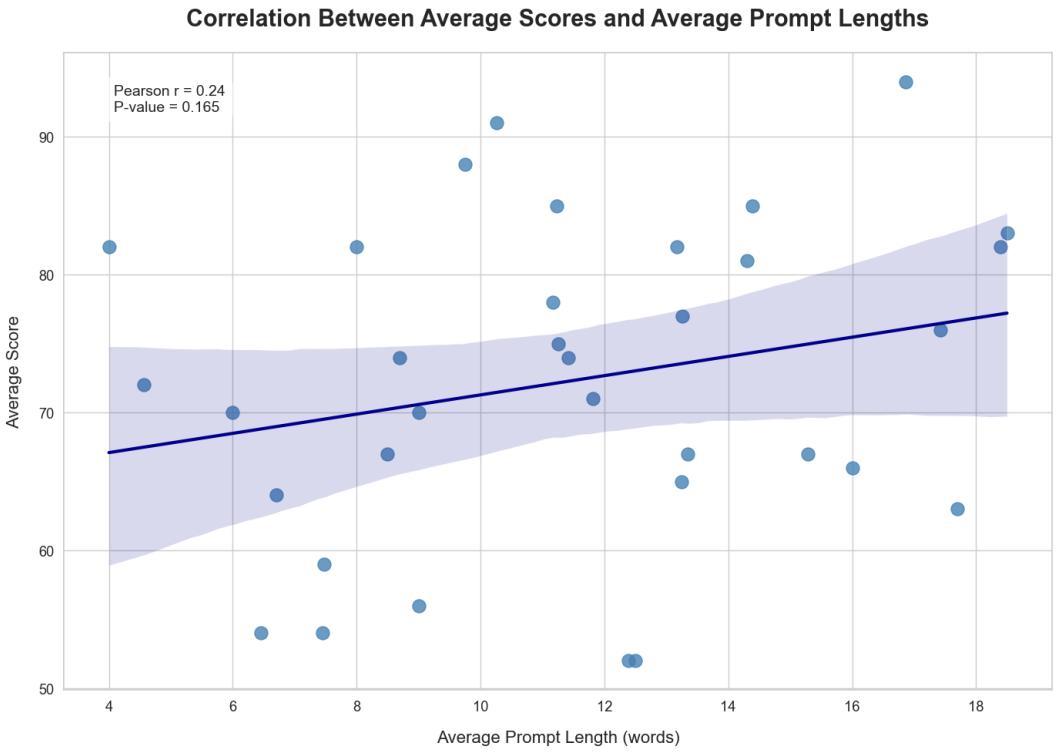


Fig. 4.8 Correlation Between Average Scores and Average Prompt Lengths.

Average prompt length increased between simple (M=12.7 words, SD=8.1) and complex tasks (M=13.3 words, SD=8.1). Optimal results were achieved with descriptions of 15-20 words, with performance declining for both shorter and longer prompts. Figure 4.8 illustrates the correlation between average scores and prompt lengths, showing no strong linear relationship between these metrics. Technical background showed no significant effect on prompt effectiveness ($\chi^2(2) = 3.42$, $p > .05$).

System Usability

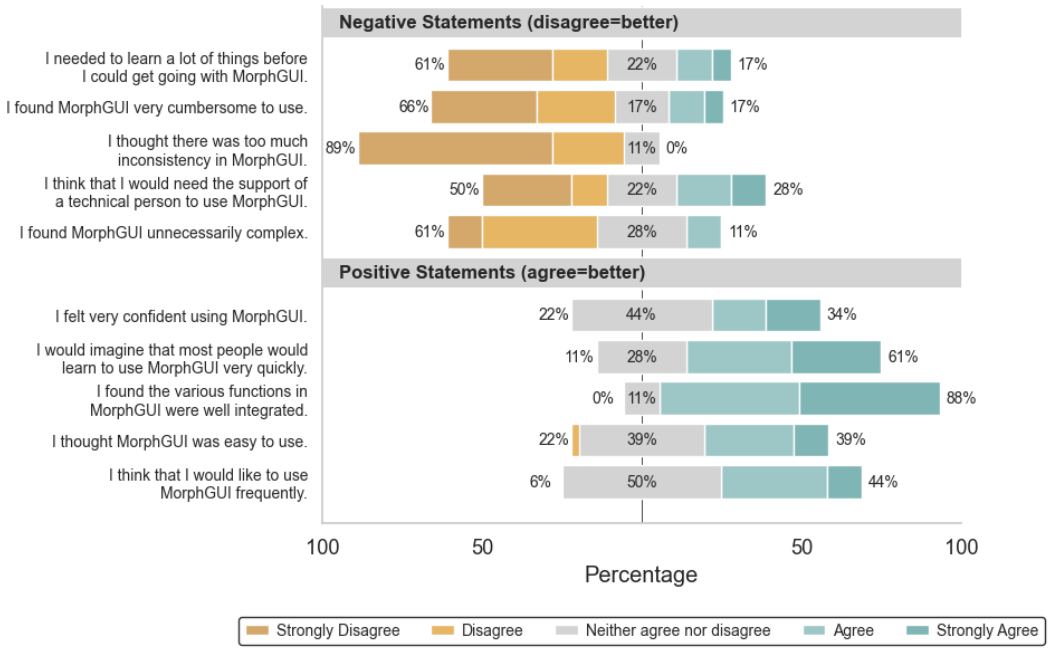


Fig. 4.9 System Usability survey results showing agreement levels with positive and negative statements about MorphGUI (N=18).

The system achieved a mean SUS score of 68 (SD=12.3), meeting the threshold for acceptable usability. Highest-rated aspects included "easy to learn" (M=4.2/5) and well integrated" (M=4.0/5). Lower scores were observed for "technical support needed" (M=2.8/5) and "system complexity" (M=2.6/5). Figure 4.9 presents detailed user feedback on various aspects of the system.

Analysis of Template Usage Patterns

Through our analysis of participants' interactions with MorphGUI's natural language interface, we identified several patterns in how users approached the separation between functionality ("What it should do") and styling/layout ("How it should appear") instructions. We used a confusion matrix approach to quantify the accuracy of template usage, as shown in Figure 4.10.

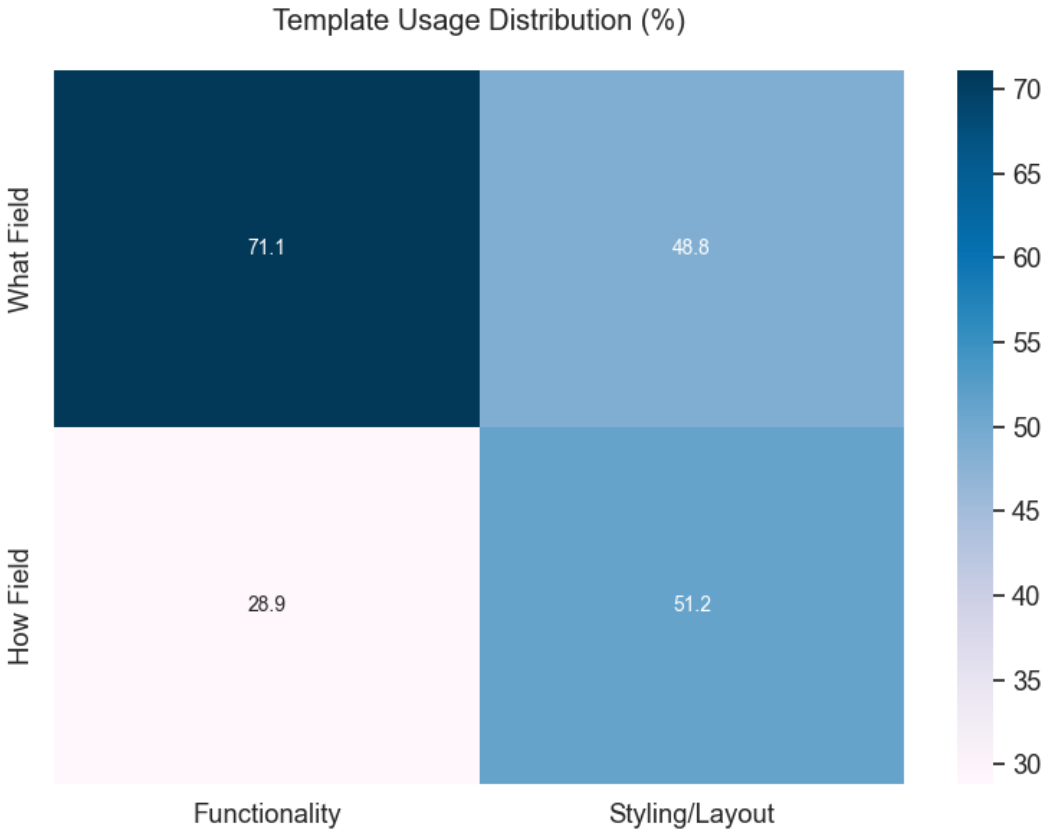


Fig. 4.10 Confusion matrix showing template usage patterns across participants. The matrix tracks: (1) True Functionality (TF): Correct placement of functionality instructions in "What it should do" field; (2) True Styling (TS): Correct placement of styling/layout instructions in "How it should appear" field; (3) False Styling (FS): Incorrect placement of styling/layout instructions in "What it should do" field; (4) False Functionality (FF): Incorrect placement of functionality instructions in "How it should appear" field.

Our analysis revealed several key patterns in template usage, as shown in Figure 4.10. On average, participants correctly placed $M = 3.5$ ($SD = 0.9$) functionality instructions in the "What it should do" field, while correctly placing $M = 8.1$

($SD = 4.8$) styling/layout instructions in the "How it should appear" field. However, we observed a consistent tendency to misplace styling and layout instructions in the "What it should do" field ($M = 7.7$, $SD = 4.7$), while functionality misplacements in the "How it should appear" field were less common ($M = 1.5$, $SD = 1.3$).

The most common error pattern was the placement of styling and layout instructions in the "What it should do" field, with participants (particularly P3, P10, and P11 with $FS > 12$) struggling with button modifications (e.g., "transform buttons to circular shape") and component positioning (e.g., "align elements to the left"). This suggests users tend to think of visual modifications as actions to be performed rather than appearance specifications.

The secondary error pattern involved functionality descriptions appearing in the "How it should appear" field, though this was less frequent. These errors typically involved repetition of functionality requirements (e.g., "should show one day at a time") rather than new functional specifications. P9 and P18 showed the highest rates of this error type (FF = 4 and FF = 5 respectively).

Notably, participants with higher rates of correct styling placement (P1, P4, P5 with $TS > 12$) generally showed lower rates of styling misplacement ($FS < 5$), suggesting that understanding the proper use of the "How it should appear" field correlates with better template usage overall.

Impact of LLM Experience

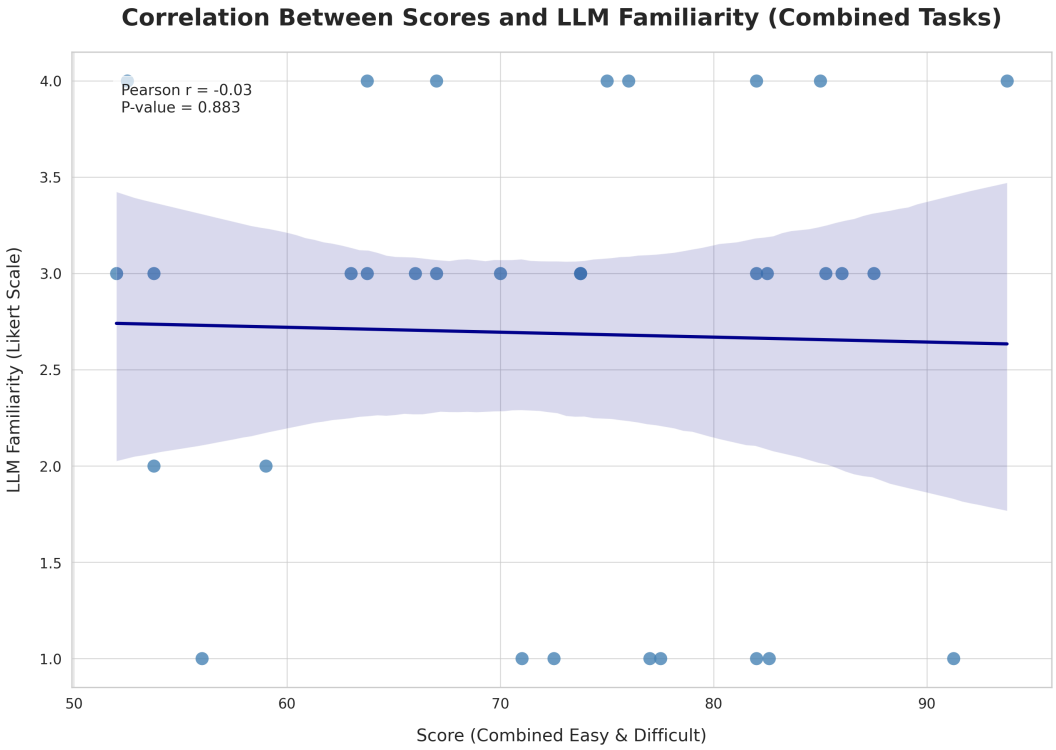


Fig. 4.11 Correlation Between Scores and LLM Usage Experience (Combined Tasks) showing no significant relationship (r = -0.03, p = 0.883)

Analysis revealed no significant correlation between participants' LLM familiarity and their task performance (r = -0.03, p = 0.883), as shown in Figure 4.11. This suggests that we cannot conclusively state whether prior experience with language models impacted or did not impact users' ability to effectively customize interfaces using MorphGUI's natural language interface.

Qualitative Feedback

Qualitative analysis of participant interviews revealed several themes that provide deeper insights into users' experiences with MorphGUI and opportunities for system enhancement. We grouped the responses by five recurring themes, highlighting both the strengths of the natural language approach and areas where additional support could enhance the user experience.

Learning Curve and Initial Experience. Most participants noted an initial adjustment period that decreased with system usage. While the interface itself was considered straightforward, users needed time to understand how to effectively formulate their customization requests. As P07 explained: "The system isn't complicated to use, but it would be useful to clarify its usage methods to understand from the start the level of specificity and complexity [...] needed in the input information to obtain the desired results." P05 similarly noted: "Initially it's more complicated to find the most comprehensive way to give instructions to the system, after various generations it became faster and easier to use."

Interface Clarity and Input Visibility. Several participants highlighted issues with input visibility and suggested improvements for interface clarity. A common concern was expressed by P14: "I would prefer to be able to view the entire text of the instructions I'm giving to the AI and to have an example of what I could write." P08 observed that "the most complicated part is understanding how to write instructions correctly for generating various parts, while understanding how to use the input system is immediate."

Request for Examples and Guidance. A recurring theme was the desire for example prompts and better guidance during the customization process. P16 stated: "the system works well, but examples with explanatory phrases could be useful to understand commands more quickly." P17 suggested to "automate the personalization process by adding examples or preset modifications to speed up the operation." P10 explicitly requested "examples before modifying something that shows how it works," while noting that "using the program often, you learn how to communicate with it."

System Effectiveness and Control. Despite initial challenges, participants generally found the system effective once familiar with it. P18 emphasized: "The system is absolutely intuitive and allows almost total control with increased use." P04 provided a nuanced observation: "The interface is quite intuitive to use even though results can vary based on how a person usually expresses themselves, and can offer a different experience for all users."

Suggestions for Improvement. Participants offered several constructive suggestions for system enhancement. P17 and P18 both suggested "highlighting the elements being modified to better understand how to proceed." P11 requested "a signal

for wrong categories when they don't appear as I would like," while P04 suggested "using keywords to standardize certain functions that allow specific actions."

These qualitative insights complement the quantitative findings presented earlier and provide valuable direction for future system improvements. The feedback particularly emphasizes the importance of providing better initial guidance while maintaining the flexibility and power of natural language interaction. The themes that emerged suggest that while MorphGUI successfully enables natural language interface customization, additional scaffolding could further improve the user experience, particularly during initial system encounters.

4.3.7 Summary

In this chapter, MorphGUI was introduced as a framework that utilizes large language models to facilitate customization of interfaces using natural language. By integrating user intent, customization directives at the component level, and dynamic code generation, MorphGUI empowers users to adjust complex interfaces without requiring specialized technical expertise. The user study illustrated that participants could effectively modify a calendar application to align with specific design objectives by articulating their desired changes through natural language commands. The findings indicate that this approach has the potential to democratize the interface personalization process, making it more accessible to a broader audience. Nevertheless, there are some aspects that warrant further investigation. Providing enhanced guidance, such as integrated example prompts or clearer instructions, could assist users in swiftly understanding the system's capabilities. Moreover, exploring the performance of this approach across various application domains, types of devices, and user demographics would help confirm its scalability and resilience. In summary, MorphGUI showcases the promise of connecting large language models with UI customization, contributing to a more comprehensive and inclusive perspective of interactive experiences. By enhancing the system to better cater to user requirements, we progress towards empowering individuals to shape their digital surroundings through natural, intuitive methods.

4.4 Conclusion

This chapter introduced two complementary approaches that leverage Large Language Models to democratize web development and customization through natural language, addressing the significant technical barriers faced by non-technical users.

The LLM-driven web development method enables users to create websites through natural language without programming knowledge, employing a template-based prompting strategy that constrains model outputs for consistent and modifiable code generation. Key innovations include an efficient modification strategy that updates only specific code sections rather than regenerating entire documents, multi-page support with linking capabilities, and iterative refinement that maintains context across modifications. Through proof-of-concept implementation using GPT-4, we demonstrated how this approach enables end users to express their requirements conversationally and refine generated websites without understanding the underlying code.

MorphGUI complements website creation by providing a framework for customizing existing interfaces through structured natural language input. By separating functional requirements (what it should do") from visual specifications (how it should appear"), the system guides users through the customization process while maintaining interface coherence. Our user study with 18 participants demonstrated that users could successfully accomplish both simple and complex customization tasks with high completion rates (74.7% and 70.2% respectively), while the system achieved acceptable usability scores (SUS=68). Analysis of template usage patterns revealed that users tend to conceptualize visual modifications as actions rather than appearance specifications, suggesting opportunities for improved guidance in future iterations. Together, these approaches demonstrate how Generative AI can transform web development and customization from technical activities requiring specialized knowledge into accessible creative processes driven by natural language. Through user studies and technical evaluations, results show that designing AI collaboration systems which allow humans to effectively express their intentions through specific requirements, and aligning the AI realization with these expressions, leads to significant improvements in user satisfaction, efficiency, and output quality compared to traditional systems. By enabling users to concentrate on their conceptual goals rather than implementation details, these approaches bridge the gap between human

creativity and technical realization, expanding who can participate in creating and customizing web interfaces.

Chapter 5

Intuitive Home Control: Disambiguation Methods for Everyday Users

Publication notice. The multimodal disambiguation framework described in this chapter was published as "Enhancing Smart Home Interaction through Multimodal Command Disambiguation" in *Personal and Ubiquitous Computing* (Springer, 2024).

In the field of ubiquitous computing, smart environments incorporate networks of devices and sensors embedded in various components—from light bulbs and appliances to wearables and the built environment itself. This integration enables systems to detect and respond to user needs, creating responsive and interactive living spaces [128, 129]. While smart technologies have diverse applications across building management [130], infrastructure [131], and healthcare [132], our research focuses specifically on residential smart home systems. Smart homes integrate Internet of Things (IoT) devices to observe, detect, and control various elements within the home environment, potentially enhancing quality of life, comfort, and resource efficiency [41, 133, 42]. However, the adoption of these technologies depends on several critical factors, including customizability, automation, accessibility, reliability, and low latency [134]—all of which influence users' willingness to integrate these systems into their daily lives. Traditional automation systems for smart homes have predominantly relied on explicit command structures and manual programming,

exemplified by tools like IFTTT[1], or pre-defined scenarios. This approach often forces users to adapt their communication to system capabilities rather than allowing systems to accommodate the natural variability of human language and preferences [31–33]. A significant challenge lies in interpreting commands, particularly when they contain inherent ambiguities in natural language. This difficulty becomes especially apparent with subjective requests such as "prepare the living room for a relaxing evening," which traditional systems struggle to translate into actionable environmental adjustments [34–37]. Under-specified commands, while often intuitively clear to humans, present significant challenges for smart home systems, leading to user frustration when systems cannot effectively handle complex requests that fall outside rigid command structures [37, 36, 35, 34, 33]. This communication gap represents a fundamental barrier to natural interaction with smart environments. Recent advancements in smart home technology have begun exploring the integration of Large Language Models (LLMs) to enhance system responses to user commands. Systems like Sasha [38] and SAGE [39] utilize LLMs to improve interpretation and execution of complex or vague commands. Sasha implements a decision-making pipeline where critical actions like device selection and routine checks are managed by an LLM. Similarly, SAGE incorporates personal preferences, physical grounding (knowledge of home devices and capabilities), and external grounding (awareness of contextual factors like weather) to provide more nuanced interactions, particularly for commands requiring contextual understanding. Despite these advancements, these systems primarily rely on text-based inputs, which may not fully capture users' intended meanings [31, 40]. Textual descriptions alone often prove insufficient for conveying complex or subjective concepts that might be more effectively communicated through visual representation. To address this limitation, we introduce a multimodal disambiguation approach for smart homes. Our system detects ambiguity in user commands and, when necessary, generates multiple possible interpretations presented through both textual descriptions and visual representations. Rather than immediately acting on its own interpretation, the system presents these options to the user for confirmation, allowing them to select the option that best aligns with their intent. By integrating this multimodal interaction with natural language command systems, we create a more intuitive interface that better mirrors human communication patterns. We hypothesize that this verbal and visual approach works particularly well for subjective requests like setting a room's ambience, where images can provide

[1] https://ifttt.com/

clearer guidance than text alone. To validate this approach, we conducted a study with seven participants evaluating the effectiveness, efficiency, and user satisfaction of textual versus visual disambiguation methods. This chapter details our multimodal disambiguation approach, the implementation of our system, and the results of our evaluation. We demonstrate how combining the language understanding capabilities of LLMs with intuitive visual representations can enhance interaction with smart home environments, making them more responsive to natural human communication while maintaining precise control over system actions.

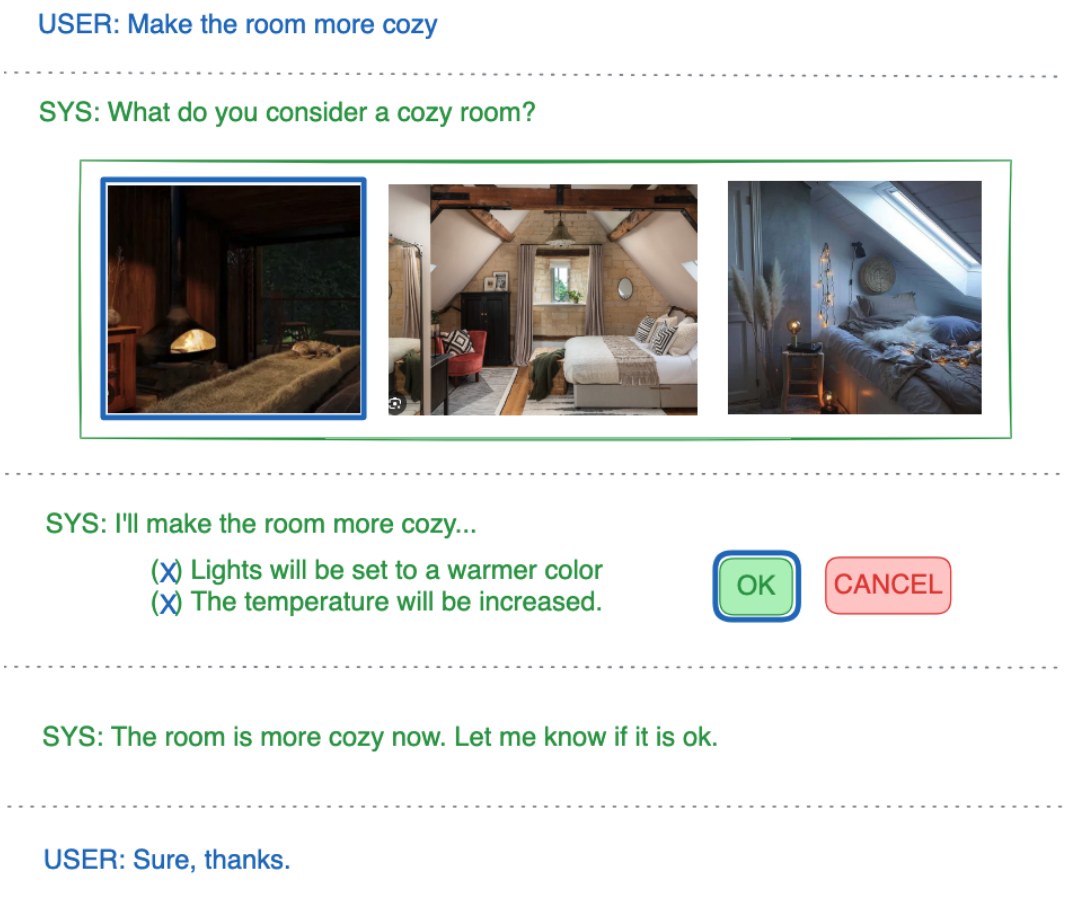


Fig. 5.1 The system records a user's command to 'Make the room cozier,' and then provides, in this instance, three visual choices to clarify the user's understanding of 'cozy.' Once the user picks their favored atmosphere using an image, the system verifies the completion of tasks such as modifying lighting and temperature to achieve the desired coziness.

5.1 Use Case for Interactive Disambiguation

In order to provide additional context and demonstrate the advantages of our methodology, we present a use case. The goal of this scenario is to showcase the disparities in user experience and system efficiency when comparing conventional text-based techniques with our recommended multimodal disambiguation strategy. Through the depiction of two situations involving identical users and contexts, we emphasize the benefits of integrating interactive disambiguation and visual prompts in the interpretation of smart-home commands, as well as the constraints present in current natural-language systems.

> *Paul, a graphic designer, comes back home from a highly busy day at the office. His living room, typically a place of peace, seems bare and unwelcoming. Wanting a tranquil environment to relax, Paul looks to his new smart home system, aiming to turn the area into a haven of serenity. He speaks the command, "Set a relaxing mood in the living room." The system, programmed to understand such directives, acknowledges the vagueness in Paul's statement. 'Relaxing' could have various meanings for different individuals – some may seek comfort in subdued lighting and gentle music, while others might favor the coziness of a simulated fireplace.*
>
> ***Utilizing Multimodal (Image and Text) Interaction:*** *The system initiates its Multimodal Concept Disambiguation procedure. It deduces that the visual modality may be appropriate for clarifying the user request and promptly generates a series of images, each illustrating a distinct interpretation of what a 'relaxing mood' might involve. One image portrays the room illuminated with soft, warm lighting accompanied by gentle music notes in the background, another displays a cozy arrangement with a virtual fireplace and ambient lighting, while a third image highlights a more natural environment with green tones and nature sounds. Paul, observing these choices on his smart TV, is immediately attracted to the image of the room featuring the virtual fireplace. It aligns with his concept of a tranquil evening - the warmth of the fire, the gentle flicker of flames, creating a calming visual and auditory sensation. He chooses this image, effectively conveying his preference without the*

necessity for elaborate descriptions. The system, after receiving Paul's selection, takes action. It adjusts the room's lighting to mimic the warm glow from the selected image, turns on the virtual fireplace on the large screen, and even subtly modifies the room's temperature to enhance the sense of warmth.

Utilizing Text-Only Interaction: *In the absence of detecting ambiguity or seeking clarification, the system lacks the necessary context to interpret Paul's concept of "relaxing." It chooses a standard action – dimming the lights and playing soft instrumental music. While this response falls within the realm of what could be perceived as relaxing, it does not completely match Paul's personal preference for the evening. Subsequently, Paul decides to enhance the directive to improve the communication of their specific desires to the system. Paul issues another directive, "Increase the warmth of the lighting and add a visual element like a fireplace." The system responds by marginally enhancing the warmth of the lighting but encounters difficulty with the abstract notion of incorporating a 'visual element like a fireplace.' It understands this as showing images of fireplaces on the smart TV in the living room rather than creating an immersive fireplace experience. Despite the room now having warmer lighting and fireplace images, it still lacks the cozy, immersive atmosphere Paul had envisioned. The system's constraints in comprehending and translating Paul's intricate request become apparent. Paul makes another attempt, refining their directive: "Replicate the lighting to imitate a fireplace's glow and play fireplace sounds." On this occasion, the system adjusts the lighting to a flickering, orange hue and plays crackling fire sound effects. While this is closer to Paul's ideal scenario, the experience still feels somewhat artificial and lacks the seamless integration of visual elements that Paul seeks.*

Throughout these iterative refinement cycles, it becomes evident that the text-only system, despite its advancements, encounters significant challenges in accurately interpreting and executing more subjective, nuanced commands. Each refinement brings Paul closer to their desired outcome, yet the process is time-consuming and somewhat frustrating, emphasizing the system's limitations in understanding and fulfilling the complete range of human preferences without additional, more

specific input. This series of refinements underscores a crucial limitation of text-only commands: they frequently struggle to convey the depth and complexity of visual information. While Paul knows what he desires, expressing it in words that the system can precisely interpret proves to be challenging. Each iteration, although approaching the desired outcome, demands effort and precise language that may not be intuitive for all users.

In comparison, a multimodal approach that integrates visual and textual cues for disambiguation permitted Paul to directly choose an image that encapsulates his concept of a serene environment. Visuals can communicate nuances like color, intensity, motion, and ambiance more promptly and comprehensively than text. This not only could have saved time but also removed the need for guesswork and iterative refinement inherent in text-only commands, resulting in a more efficient and gratifying user experience.

5.2 Method

Upon completion of the figure display, the proposed multimodal disambiguation system's architecture, as illustrated in Figure 5.2, employs a unified and flexible methodology for understanding and reacting to user directives. By possessing the ability to learn and adjust iteratively, the system presents a smart-home framework that comprehends and evolves alongside the user's distinct preferences and behaviors.

Context Store: The Context Store acts as a central storage unit within the smart home AI infrastructure, storing data obtained from user engagements and environmental information. It is maintained and updated through an LLM, which is designed to process and enhance the information via structured inquiries. To provide technical clarity, the Context Store is implemented as a versioned JSON object. This format allows for a structured representation of the home environment, device states, and user preferences. An example structure is as follows:

```
{
  "version": 3,
  "home_layout": {
    "living_room": ["light_1", "thermostat_1"],
    "kitchen": ["light_2", "smart_plug_1"]
```

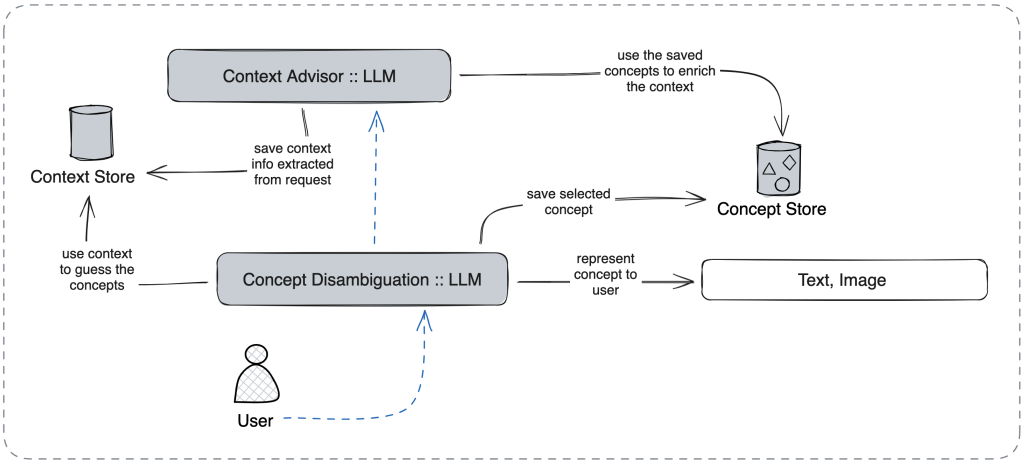


Fig. 5.2 The diagram illustrates the process in which the AI interacts with a user to resolve ambiguous instructions in a smart home scenario. The AI receives input from the user and the context store, seeks advice from the concept advisor to generate a concept based on the environment, and then stores this concept. The Concept Disambiguation module utilizes LLMs to offer the user different modalities (text or image) to represent the concept, from which the user chooses for the AI to act upon, thereby completing the feedback loop of comprehension and action within the smart home environment.

```
  },
  "devices": {
    "light_1": { "type": "hue", "state": "on", "brightness": 80 },
    "thermostat_1": { "type": "nest", "state": "on", "temp_c": 21 }
  },
  "user_history": [ ... ]
}
```

When a user command leads to a state change, the backend generates a new, versioned JSON object representing the updated context. This object is then sent to the frontend, which triggers a re-render to ensure the UI always reflects the current state of the smart home environment.

Specifically, the suggested system incorporates multiple interconnected components for interpreting, clarifying, and executing user directives, which include:

Context Advisor: Leveraging the capabilities of LLM, the Context Advisor suggests ideas that closely match the user's surroundings and requests by utilizing data sourced from the Context Store. Functioning as a pivotal intermediary, the

Context Advisor transforms extensive contextual data into practical insights. This process is vital for merging the context with the disambiguation results, guaranteeing that the system's reactions and operations are pertinent and tailored to the individual user's requirements and inclinations. The LLM evaluates the context based on the user's directions to ensure that the AI's comprehension aligns with the user's factual conditions and preferences. This interplay between the Context Store and the Context Advisor is crafted to be flexible, enabling the system to adjust to the user's evolving needs and enhance the context over time. Structured inquiries are employed to extract data from the Context Store, resulting in a more context-specific knowledge.

Over time, as the user engages with the system and their living environment changes, the data in the Context Store is continually updated. This strategy is crucial for maintaining a responsive smart home system that is sensitive to the user's needs, offering personalized responses and actions that are relevant to the user's requirements and present circumstances.

Concept Store: The Concept Store, distinct from the Context Store, is focused on storing user-selected interpretations and their representations, essentially creating a 'memory' of user preferences. While the Context Store deals with environmental data, the Concept Store plays a significant role in modeling user interactions and preferences, ensuring that future system responses closely match the user's past choices and expectations. This distinction enhances the system's ability to predict and respond accurately to user commands.

Concept Disambiguation: The Concept Disambiguation subsystem of the smart-home AI utilizes an LLM to clarify ambiguous user commands. It is the central component of the proposed method, positioned closer to the user. To resolve ambiguities, the subsystem follows a multi-step process: beginning with a prompt designed with a dataset of ambiguous commands tailored for smart homes, the system categorizes these commands based on their levels of ambiguity. For commands where visual cues can aid in clarification, the system generates unique, non-overlapping visual cue descriptions and corresponding AI-generated images. Users then choose the image that best matches their intention, enabling the Context Advisor to create a customized policy for home automation. The intricate details of this process will be further explained in the subsequent.

5.3 Concept Clarification

The subsystem for Concept Clarification is central to our smart-home AI, utilizing Large Language Models (LLMs) to clarify human commands. By understanding user intent, it ensures that every instruction is fully understood and translated into actions that align with the user's true intentions. It comprises five sub-components and mechanisms that provide this understanding to the system:

Multimodal Selection for Resolving Ambiguity We compiled a dataset of 55 real-world smart home command instances, covering both ambiguous directives like "establish a soothing atmosphere in the kitchen" and straightforward commands like "turn on the kitchen lights." Each instance is tagged with potential ambiguity indicators and the recommended mode for clarification (text, image). To test zero-shot inference, we examined the capability of a large language model to classify the commands as ambiguous or unambiguous without any prior exposure to this dataset. Despite lacking training, the model achieved 70% accuracy in line with human assessments, showcasing reliable detection of ambiguity even in unfamiliar directives. This dataset will facilitate further exploration into multimodal clarification systems. To enhance the smart-home system's capacity to handle ambiguous directives, a crucial step involves evaluating each directive in the dataset to determine if visual cues can effectively resolve ambiguities. For every directive identified as ambiguous in our dataset, an additional assessment is carried out to ascertain if visual cues are a suitable approach for clarification. Indeed, certain ambiguities may be more effectively resolved through additional textual details or alternative methods. This strategy aims to utilize the most suitable mode for each specific scenario, thereby improving the system's responsiveness to user directives in an optimal manner.

Creation of Non-Overlapping Visual Cue Captions When a command in our dataset is identified as needing visual clarification, our system follows a procedure to produce unique, non-overlapping visual cue captions. This process aims to present various interpretations for the ambiguous concepts. We employ a Large Language Model (LLM) to generate captions that provide distinct interpretations. To assess the LLM's ability to generate diverse concepts, we compared the generated captions with those from free generation, using metrics in the embedding space. This comparison helps determine the extent of semantic variation among the captions. The embedding

space metrics offer a quantitative evaluation of the diversity in the concepts generated by the LLM, confirming that the captions capture different facets of the ambiguity.

User Selection and Policy Generation Presented with images or textual descriptions, the user participates in the final act of selection by identifying the depiction that most closely aligns with their intent. The Context Advisor subsequently creates a policy that integrates the user's selection, blending the chosen interpretation with the smart home's contextual data. This individualized policy guides the home automation system, guaranteeing that the user's initial ambiguous command translates into a result that aligns with their expectations. Through this approach, our system not only resolves uncertainties, but accomplishes this by involving the user, gaining insights from their decisions, and consistently enhancing its grasp of their preferences.

5.4 Implementation

The multimodal disambiguation system was developed utilizing a Python server backend and React frontend interface.

We utilize the OpenAI API for accessing GPT-4[2], to classify commands as ambiguous or not in a zero-shot scenario. The components outlined in our design are coordinated using the LangChain[3] framework. Distinct captions are produced using GPT-4 and assessed using embedding space metrics to confirm diversity. Utilizing the captions, we utilize DALL·E 3[4] to create corresponding images illustrating potential visual disambiguation choices. The setup involves a React frontend[5] enabling users to choose the textual or visual option that best aligns with their intention. The system as a whole is structured for flexibility. As newer language or generative models become available, they can be seamlessly integrated to improve disambiguation accuracy. We aim to make key components open-source to facilitate further research.

[2] https://openai.com/api/
[3] https://github.com/LangChain/langchain
[4] https://openai.com/dall-e
[5] https://react.dev

5.5 User Study

A user study was carried out to assess the efficacy of the disambiguation method and the appropriateness of the visual and textual representation produced in smart homes. Seven participants with diverse backgrounds were involved in the study. The evaluation included both quantitative and qualitative assessments. Quantitatively, participants provided ratings on the system's responses regarding intent alignment and effectiveness.

Qualitatively, participants shared their feedback and opinions following interactions with the system. They deliberated on their preferences between textual and visual outputs, the system's ease of use, and their overall satisfaction with its performance. This feedback provided deeper insights into the user experience, emphasizing the practical implications of the system's performance and areas for enhancement.

5.5.1 Procedure

Prior to the experiment, participants were evaluated for their familiarity with smart home systems, including any prior interactions with voice or text-based assistants such as Alexa or Google Assistant. They were queried about their expectations regarding usability, responsiveness, and accuracy in these systems, as well as their communication preferences with smart devices.

Subsequently, participants were presented with a sequence of vague commands in natural language for execution. Examples of these commands included, for instance, "Create a relaxing ambiance when I arrive home," "Turn on the lights if a child enters the house," or "Set romantic lights in the kitchen." Following each command, participants were provided with options to disambiguate, involving the use of either images or textual descriptions, and the system's response was assessed based on how closely it matched their expectations.

To ensure a fair comparison between the two modalities, the sequence in which the instruction modes were presented was counterbalanced across participants.

During the post-experiment phase, participants reflected on the system's responses and deliberated on the efficacy of the modality they utilized. They juxtaposed their experiences with textual and image-based responses, expressed their perspec-

tives on usability, and put forward suggestions for enhancing future interactions with smart home systems.

5.5.2 Participants

#	Age	Occupation	Gender	Familiarity with Smart Home Systems
P1	24	Early childhood educator	Female	Very little, used specifically rather than in daily activities.
P2	27	Software engineer	Male	Limited to Smart TV and phone assistant, not used for house control.
P3	25	Photographer	Female	Aware of systems but no personal use or knowledge.
P4	29	Management assistant	Female	Knowledgeable about multiple functionalities, used Alexa for music.
P5	23	Student	Male	Limited familiarity, used Google Home for music.
P6	34	Early childhood assistant	Female	Limited familiarity, used Google Home for music.
P7	28	Software engineer	Male	Familiar with IoT aspects of smart home systems.

Table 5.1 User study's participants

Participants were recruited through convenience and snowball sampling methods by sending private messages to social circles. To ensure a balanced population, potential participants were asked to complete a demographic survey to minimize self-selection bias. A total of seven participants were selected, representing a diverse demographic in terms of age, gender, and familiarity with smart home systems. Prior to the study, participants signed an informed consent form.

The age range of the participants ranged from 23 to 34 years. Gender distribution included both male and female participants to ensure a balanced representation. The study was conducted in Spanish, the native language of all participants.

Participants showed a wide range of familiarity with smart home systems. Some had limited interaction, using smart home systems for specific tasks rather than in their daily routines, while others were more experienced and regularly used devices

such as Smart TVs, phone assistants, and features like Alexa for music. This diversity in familiarity levels provided valuable insights into user interactions with smart home systems during the experiment.

Table 5.1 summarizes the participant profiles, offering a concise overview of their demographics and experience with smart home systems.

5.6 Results

5.6.1 Analysis of Quantitative Results

The study sought to evaluate the correspondence between the system's reaction and the user's intention, as well as the effectiveness of each mode.

Users evaluated the system's reactions on a scale ranging from 1 to 5, where 1 indicated the lowest effectiveness and 5 indicated the highest, in relation to *intent alignment* and *efficacy*. Intent alignment refers to the extent to which the system's disambiguation matched the user's genuine intention, while efficacy relates to the overall effectiveness and usefulness of the response.

The analysis showed that in the visual modality, the mean intent alignment score was $\mathbf{4.07} \pm 0.73$, and the mean efficacy score was approximately $\mathbf{3.43} \pm 1.28$. On the other hand, the textual modality had an average intent alignment score of $\mathbf{4.00} \pm 1.04$ and an average efficacy score of $\mathbf{3.86} \pm 1.29$. The results, as depicted in Figure 5.3, reveal slight differences in both intent alignment and efficacy between the two modalities. Although the visual modality displayed slightly higher intent alignment, the textual modality exhibited slightly greater efficacy. Nevertheless, these variances are not significant, indicating that neither modality consistently surpasses the other in these aspects. The study noted that both modalities were generally effective, with minor discrepancies in specific scenarios. The textual modality was praised for its straightforwardness and clarity, while the visual modality offered a more instinctive and visual approach to interacting with the smart home system.

This outcome implies that the selection between textual and visual modalities may rely more on the specific context and user preference rather than a clear advantage of one over the other. For example, the textual modality, with its slightly higher efficacy,

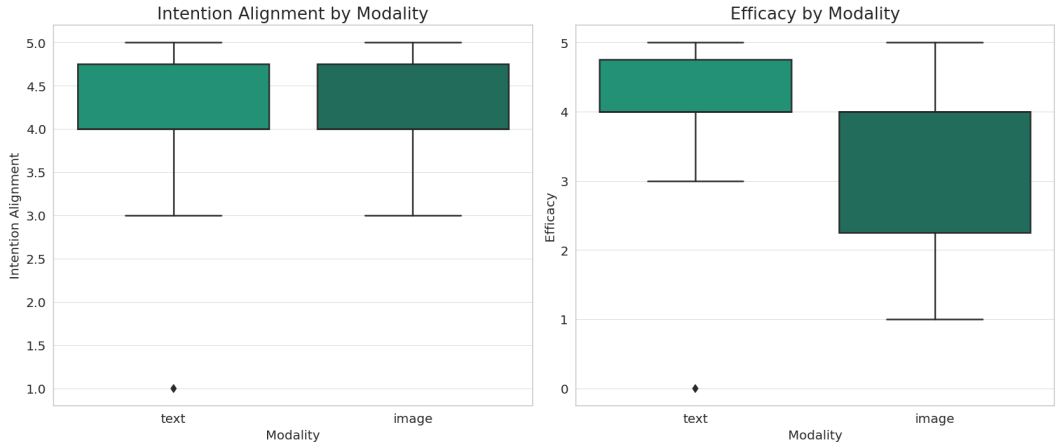


Fig. 5.3 Alignment of intentions and efficacy for disambiguation results using both text and images.

could be more appropriate for situations where precision and explicit instructions are crucial.

On the flip side, the visual modality, which exhibited slightly higher intent alignment, may be more efficient in scenarios necessitating a more comprehensive or subjective understanding, like creating a specific atmosphere or setting a mood in a space. These results endorse the notion that in situations where the objective is more about establishing a vibe or an experience rather than carrying out a precise task, the visual cues provided by the visual modality can offer a more intuitive and thorough grasp of the user's intent. Additionally, this modality plays a crucial role in clarifying concepts that may not be easily conveyed or articulated through plain language. For example, communicating the idea of "coziness" or "relaxation" can be more effectively accomplished through visual means rather than text, as these concepts can have diverse interpretations that are more effectively captured visually. This capacity to bridge the communication gap where language may be inadequate renders the visual modality a valuable asset in improving user interaction with smart home systems.

5.6.2 Evaluation of Qualitative Findings

Consistency with User Intentions To guarantee user contentment with the disambiguation system, it is imperative that the system's selections closely align with the

users' true intentions. After conducting our experiment, we carried out interviews to assess if the system's disambiguation accurately mirrored users' underlying intent.

Generally, participants expressed a high degree of satisfaction with the system's alignment with their instructions. Participant 1 (P1) encapsulated this viewpoint, observing that while the responses were typically satisfactory, there were instances where further clarification was needed. This suggests an overall favorable interaction with the system, albeit with opportunities for enhancing understanding of user intent.

Issues of accuracy and specificity were highlighted. For instance, Participant 3 (P3) remarked that the system's responses were occasionally lacking in specificity: "I think the system's response was good on average but it wasn't specific enough with some of the commands" (P3).

Concerns were also raised about the system's ability to accommodate varying perspectives. P1 noted the system's shortcomings in representing different yet conceptually similar situations, using temperature as an illustration, "Ineffective that it does not vary perception, vibrating temperature is 21C, comfortable temperature is 21C" (P1). This feedback emphasizes the necessity for context-aware responses from the system, capable of distinguishing subtle discrepancies in user instructions.

Effectiveness of Responses The participants' experiences also shed light on the balance between automated responses and user expectations. While the system generally aligned with user intentions, it occasionally required further input or correction from the user to fully understand the intended command. This aspect of interaction indicates a need for continuous learning and adaptation in smart home systems to ensure they evolve in better understanding and anticipating user needs.

While the alignment with user intentions focused on the system's ability to match the user's true intention after the disambiguation process, the system's response effectiveness was evaluated based on how well it executed user commands.

Participant 2 praised the system for its prompt response time and user-friendly interface; the ease of selecting options significantly influenced their positive evaluation, stating, "*One of the particular aspects that I consider effective is the response time and the intuitiveness to select any option*".

Participant 4 offered a more nuanced perspective, recognizing the system's skill in integrating control across multiple devices, which improved the overall effectiveness

in establishing a unified environment: "*Mixing multiple devices even if they were not in the same room was effective*". However, Participant 4 also observed the presence of options that appeared unrealistic or impractical, suggesting that while the system excelled in complex integration, it sometimes struggled to provide realistic or feasible solutions.

These viewpoints underscored that the effectiveness of smart home systems relies on the responsiveness and intuitiveness of the interface, the system's contextual awareness and precision in representing available features, and the practicality of the options it offers.

Ease of Use The discussion on user-friendliness indicated that participants generally perceived the system's modalities as easy to use, although their preferences differed depending on the task and individual inclinations.

Participant 6 (P6) emphasized the intuitive aspect of both visual and textual modalities. They noted that the combination of these modalities helped in reaching a more precise answer, suggesting that integrating various forms of information presentation could improve user comprehension and system efficiency. This perspective emphasizes the advantages of multimodal interfaces in smart home systems, providing users with a more comprehensive and accessible means of engaging with the technology.

Nevertheless, user preferences concerning modalities varied. Participant 1 (P1) expressed a partial preference for images, acknowledging their general usefulness while noting that they may not always capture all aspects of their thoughts. This observation underscores the limitations of visual representations in conveying complex or abstract concepts, which are crucial in situations requiring precision and thoroughness.

On the contrary, Participant 4 (P4) exhibited a distinct preference for text, citing its capacity to articulate intricate ideas that images may struggle to interpret effectively. This preference underscores the strengths of textual information in delivering detailed and specific instructions or descriptions, which play a vital role in ensuring accurate system responses, especially in complex scenarios.

The research demonstrates a varied landscape of user preferences and perceptions concerning the usability of smart home system modalities. While there is a general

trend favoring the intuitive nature of images and text, individual preferences differ depending on the context and information conveyed. This diversity highlights the significance of customizable and adaptable smart home systems that can meet various user needs and preferences by offering different modalities or a combination of them to improve user experience and system efficiency.

Comparative Perception Through a Comparative Perception evaluation, insights were obtained on users' preferences between textual and visual modalities, revealing differences in how individuals resolve ambiguities. Participant 2 (P2) expressed a stronger preference for the text modality, attributing it to the specificity and clarity it offers. They indicated that textual information provided a more detailed understanding of the system's functionality, stating, "Even though I consider myself a visual reception person, I felt more comfortable with the text modality because it was more specific and, in my opinion, it was clearer what the system was going to do." This preference underscores the importance of precise and comprehensive communication in smart home systems, especially in understanding user command implications.

In contrast, P3 held the belief that images were more effective, particularly for users who find visual examples easier to understand than textual descriptions. This viewpoint is in line with the concept that visual representations can simplify complex ideas, making them more accessible and easier to grasp for specific users "I think images are more effective because for some people it's easier to understand something if shown visual examples than text that could be interpreted in different ways." P3's perspective emphasizes the potential of images to bridge gaps in comprehension, especially in situations where text may have varying interpretations.

These differing opinions demonstrate the diversity in user preferences and cognitive styles. While P2's inclination towards text over images, despite being visually oriented, demonstrates a nuanced approach to information processing, P3's focus on the effectiveness of images for certain users highlights the broad appeal and accessibility of visual representations.

User preferences for modalities are not always straightforward and can be influenced by factors beyond primary learning styles. This implies that smart home systems should be adaptable and versatile, providing both textual and visual modes to meet the varied needs and preferences of users. This adaptability could play a

crucial role in improving user experience and ensuring that interactions with the system are intuitive, efficient, and tailored to individual user preferences.

Recommendations and Enhancements The discussion yielded a range of valuable suggestions from participants, reflecting their expectations from the smart home system.

P1 suggested a hybrid approach, recommending the integration of textual descriptions with images. This proposal is particularly insightful as it addresses the limitations of both modalities when used independently. By combining images with descriptions, the system can provide both the clarity and specificity of text with the intuitive appeal of visuals: "*Images should come with a description, the best of both worlds*." This strategy could be especially advantageous in situations where complex ideas or instructions need to be communicated, ensuring that users gain a comprehensive understanding of the system's responses while retaining the ease of disambiguation with visual cues.

P5 concentrated on the aesthetic aspect of images, highlighting the significance of their visual appeal and accuracy in representation. P5's suggestion underscores the importance of aesthetic design in user experience, where visually appealing images can enhance engagement and satisfaction. Furthermore, ensuring that images accurately depict the intended concept or action can prevent misunderstandings and improve the system's effectiveness: "*Put images that are pleasing to the eye and that do not alter perception*."

P7 emphasized the necessity for a wider variety of options in image-based responses. By expanding the pool of images available for selection, the system can provide a more nuanced array of choices, catering to diverse user preferences and scenarios. Allowing users to choose multiple images could facilitate a more detailed and personalized interaction with the system, as users could combine different elements to better suit their needs: "*Expand the number of images, allowing you to select a maximum of two images at a time*."

The recommendations provided by participants highlight a distinct need for a smarter, more user-friendly, and visually appealing interface in smart home systems. The enhancement of textual descriptions alongside images, improvement in image quality, and expansion of selectable options have been identified as crucial areas for enhancement. The implementation of these modifications has the potential to greatly

increase user satisfaction and enhance the system's overall effectiveness, resulting in improved adaptability and responsiveness to individual user requirements.

5.7 Conclusions

As smart home systems become more prevalent, there is an increasing demand for more user-friendly interaction methods. Traditional natural-language systems often struggle to understand ambiguous, subtle, and subjective commands, resulting in user dissatisfaction and suboptimal outcomes. Our proposed multimodal disambiguation approach tackles these challenges by integrating a system with ambiguity detection, separate textual and image generation, and user-driven concept selection between visual or textual representations. Quantitative and qualitative analysis from a user study showed overall effectiveness while also identifying areas for improvement.

However, the study also revealed important limitations and areas for future work. A key challenge is the system's reliance on visual representation for disambiguation, which is less effective for intangible or non-visual attributes. Concepts like temperature, humidity, or abstract states like "security level" lack a clear, singular visual metaphor. While the system excels at representing ambiance (lighting, colors), future work must explore hybrid representations, such as overlaying textual data or icons (e.g., a thermometer with "21°C") onto generated images to effectively communicate these intangible states.

Furthermore, the current system is reactive, responding only to direct commands. A significant extension would be to support proactive, rule-based conditions (e.g., "when I arrive home after 8 PM, make the living room cozy"). This would require enhancing the natural language processing to parse conditional logic and implementing a new module to store and trigger these rules based on contextual cues like time or user location. Such an extension would transform the system from a simple command interpreter into a more sophisticated home automation engine.

Finally, it is crucial to analyze scenarios where visual disambiguation may be unnecessary or even counterproductive. For simple, unambiguous commands ("turn on the kitchen light") or for expert users who can issue precise instructions, the disambiguation step introduces unneeded friction. Future iterations should incorporate a confidence score for the initial command interpretation; only when

the confidence is low would the visual disambiguation be triggered. The system should also always provide a manual override, allowing users to bypass the visual selection and enforce a direct command, ensuring that the system enhances, rather than hinders, user efficiency.

Expanding the command dataset and disambiguation options could provide a wider range of scenarios and choices. Fine-tuning through further real-world testing would enhance components such as the caption generation model and the display of the multimodal interface. We need to tackle the difficulties of conveying complex ideas through each modality and incorporating choice selection with home automation policies. Ultimately, future systems should prioritize adaptability to various environments, command structures, user requirements, and preferences. In summary, we have showcased the potential of ambiguity detection and multimodal interaction techniques that merge textual and visual cues to enhance the usability and precision of smart home control. We anticipate that our research will encourage broader adoption and community advancement in this intersection of AI, HCI, and ubiquitous computing.

Chapter 6

Discussion and Future Work

This dissertation has presented several novel approaches to lowering technical barriers for creating and customizing interactive systems through Generative AI. This chapter discusses common threads and patterns across these domains, examines limitations of our approaches, and outlines promising directions for future research.

6.1 Common Themes and Design Principles

Across our studies in the distinct domains, clear patterns emerged in how Generative AI transforms creation processes. We identified several principles that transcend specific applications and point toward a more general framework for AI-assisted creation and customization.

Bridging Domain Knowledge and Technical Implementation Our research reveals a critical insight for bridging domain expertise and technical implementation: AI systems must not only substitute for the knowledge users lack but must structure the control flow to mirror how genuine experts in that domain would work. This dual requirement—providing missing expertise while preserving authentic expert workflows—ensures users maintain meaningful control throughout the creation process.

When educators use our tutoring system tools, the AI does not merely generate interfaces based on pedagogical requirements; it structures the decision-making

process to follow how instructional designers actually work—moving from learning objectives to content scaffolding to interaction design. Similarly, our design-to-code systems preserve the natural progression that web developers follow, from structure to style to interaction, rather than forcing designers to adapt to artificial technical sequences.

This mirroring of expert workflows proves crucial for effective knowledge substitution because it allows domain experts to apply their judgment at the same intervention points where professionals in the missing knowledge domain would make critical decisions. By structuring AI assistance to follow authentic expert processes rather than technical conveniences, users can stay firmly grounded in familiar territory while the system handles unfamiliar aspects.

The effectiveness of this approach manifests in how users maintain agency despite significant knowledge gaps. When the control flow aligns with real expert practices, users can make informed decisions by applying their existing domain knowledge to each step, even when the technical implementation details remain hidden. This alignment between system flow and expert practice creates a natural scaffold that extends users' capabilities while preserving their sense of control and ownership over the final product.

Balancing Automation and Control A consistent theme across our contributions is the practical balance between automation and user control. Our approach scaffolds the creation process through step-by-step guidance rather than attempting full automation, preserving user agency while removing technical barriers.

In our MorphGUI system, this balance manifests through structured templates that guide users' natural language customization requests while providing real-time visual feedback. Users maintain control over desired outcomes but do not need to understand underlying implementation details. Similarly, our Design2Code approach enables designers to rapidly prototype functional interfaces by automatically translating visual designs to code, while preserving their creative control through style references and sketch annotations for dynamic behaviors.

For educators creating tutoring systems, our decomposition of high-level goals into explicit pedagogical steps creates a scaffolded creation process where AI handles technical implementation while educators maintain control over instructional

strategy. Rather than forcing adaptation to technical constraints, the system adapts to educators' pedagogical intentions through preference-driven refinement.

This scaffolded approach to balancing automation with control offers practical advantages over both fully automated systems and traditional technical tools. By providing immediate visual feedback during the creation process, users can iteratively refine their intentions without switching contexts between design and implementation.

The result is a new interaction paradigm where domain experts can focus on their areas of expertise—whether pedagogical strategy, visual design, or desired functionality—while the system handles the technical translation into functional implementations.

Multiple Pathways to Expression A significant finding across our research is the value of providing multiple modalities for expressing intentions, each aligned with the natural cognitive frameworks and domain expertise of different user groups. Traditional development environments impose a uniform, technically-oriented expression method—typically code writing—which forces users to translate their domain-specific mental models into programming constructs.

For educators creating tutoring systems, we observed that their expertise manifests primarily through instructional sequencing and pedagogical strategy rather than interface specifications. By providing visual selection of design alternatives instead of requiring written specifications, we enabled a form of expression that aligns with educators' evaluative expertise—their ability to recognize effective teaching interfaces even if they couldn't articulate formal design principles. This visual selection pathway preserves the nuanced pedagogical judgment educators have developed through practice while bypassing the technical design vocabulary they typically lack.

For designers, we enabled a multi-layered expression approach through sketches (for structural elements), reference images (for style attributes), and symbolic annotations (for interactive behaviors). This separation of concerns allows designers to leverage their existing visual thinking processes and established design languages without requiring them to mentally translate these concepts into programming constructs. The separation proved particularly valuable for exploring design alternatives, as designers could rapidly iterate on different aspects of the interface indepen-

dently—changing structural layouts without disrupting style choices, or modifying interactive behaviors while maintaining visual consistency.

For end-users customizing applications, we facilitated natural language expression structured through guided templates. This approach acknowledges that while end-users may not possess technical vocabulary, they have sophisticated models of their own needs and preferences that can be articulated conversationally. The structured guidance serves as a scaffolding that helps bridge between everyday language and system capabilities without requiring users to learn a formal specification language.

In smart home environments, our research revealed the limitations of purely textual communication for subjective concepts. By supplementing language with visual disambiguation options, we discovered that certain qualities are more efficiently communicated through exemplars than descriptions. This visual pathway allows users to externalize aesthetic and emotional intentions that would otherwise require extensive verbal elaboration.

The cognitive implications of allowing users to express intentions through modalities aligned with their expertise, benefits the preservation of domain complexity and enables a more democratic technological landscape where diverse forms of expertise can directly shape digital artifacts. Unlike simplified "beginner" interfaces that often flatten rich domain knowledge into limited options, our multi-pathway approach maintains the sophisticated nuances of different expertise areas. Educators can apply complex pedagogical principles, designers can implement subtle aesthetic judgments, and end-users can express contextual preferences—all without these being reduced to the lowest common denominator of what's easily expressible in code.

6.2 Limitations and Challenges

While our research demonstrates significant promise for Generative AI in lowering technical barriers, several important limitations and challenges remain to be addressed.

Model Capabilities and Knowledge Boundaries Current Generative AI models, including those employed in our research, have inherent limitations in their knowl-

edge and capabilities. They may generate incorrect or outdated code, misunderstand domain-specific terminology, or produce outputs that appear plausible but contain subtle errors. These limitations are particularly challenging in technical domains where precision is essential. While our approaches incorporate various guardrails and feedback mechanisms to mitigate these issues, the fundamental limitations of underlying models remain a significant constraint. Future improvements in model capabilities, domain-specific fine-tuning, and retrieval-augmented generation could help address these limitations, but complete elimination of errors is unlikely in the near term. This suggests a continued need for human oversight and verification, particularly for critical applications.

Explainability and Transparency Many of our approaches use complex Generative AI models that lack transparency in their reasoning processes. This "black box" nature can make it difficult for users to understand why particular outputs were generated or how to effectively guide the system toward desired outcomes. While our approaches provide user control, they do not fully address the explainability challenge. Users may still struggle to understand model limitations or effectively diagnose and correct issues when outputs do not meet their expectations. Future research should explore methods for improving model explainability and providing users with clearer insights into the generation process. This might include exposing confidence scores, highlighting reference sources, or providing alternative generations with explanations of their differences.

Evaluation Challenges Evaluating Generative AI systems for interactive system creation presents unique challenges. Traditional metrics like accuracy or error rates are insufficient when outputs have subjective quality dimensions and must satisfy both functional requirements and user preferences. Our user studies provide valuable insights into effectiveness, efficiency, and satisfaction. However, it is important to explicitly acknowledge that these studies were conducted with small sample sizes, which limits the statistical power and generalizability of our findings. While insightful, the results should be considered preliminary. Future work must involve larger and more diverse participant pools to validate these initial findings. Furthermore, longitudinal studies would be necessary to assess long-term adoption and impact. Additionally, it remains challenging to isolate the effects of different system components or compare approaches on a level playing field given their different target

users and contexts. Future research should develop more comprehensive evaluation frameworks specifically designed for Generative AI systems in interactive contexts, incorporating both objective metrics and subjective assessments across multiple dimensions of quality.

Ethical Considerations The use of Generative AI in lowering technical barriers raises important ethical considerations regarding authorship, responsibility, and accessibility. As systems increasingly generate complex artifacts based on user input, questions arise about attribution, ownership, and accountability for the resulting outputs. There is also the risk that reducing technical barriers might lead to a proliferation of low-quality or problematic systems if users lack understanding of best practices in design, accessibility, or security. While democratizing creation is a worthy goal, it must be balanced with appropriate guidance and safeguards. Future research should explicitly address these ethical dimensions, developing frameworks for responsible use of Generative AI in system creation and customization, and incorporating ethical guidelines directly into system design.

6.3 Future Research Directions

Based on our findings and the limitations discussed above, several promising directions for future research emerge.

Integration Across Creation and Customization Workflows Our research addressed creation and customization as somewhat separate processes, but future work could explore deeper integration between these activities. For example, systems could preserve the design history and rationale during creation to facilitate more informed customization later, or customization patterns could feed back into creation processes to improve initial designs. This integration would better reflect the iterative nature of real-world development, where creation is rarely a one-time activity but rather an ongoing process interleaved with customization and refinement. By maintaining continuity across these processes, systems could provide more coherent support for the full lifecycle of interactive systems.

Advanced Multimodal Interaction While our research incorporated multiple modalities for expressing intentions (text, images, sketches), future work could explore more sophisticated multimodal interaction patterns. This might include combining speech and gesture, integrating physical manipulation with digital interfaces, or developing hybrid approaches that seamlessly blend different expression modes. Advances in multimodal foundation models that can process and generate content across text, image, audio, and video modalities offer particularly exciting opportunities for more natural and expressive interaction. These models could enable users to communicate their intentions through whatever combination of modalities feels most natural for a given task.

Collaborative Creation and Customization Our research primarily focused on individual users interacting with Generative AI systems, but many real-world creation and customization tasks involve collaboration among multiple stakeholders. Future research could explore how Generative AI can support collaborative processes, mediating between different perspectives and helping reconcile potentially conflicting requirements. This direction might involve developing shared workspaces where multiple users can interact with the same generative system, explicit support for role-based access and contribution, or mechanisms for negotiating and resolving differences in preferences or priorities.

Adaptation and Learning from User Behavior While our systems incorporated user feedback in various ways, they generally did not adapt their behavior based on patterns observed across multiple interactions. Future research could explore how Generative AI systems might learn from user behavior over time, gradually adapting to individual preferences and working styles. This adaptation might involve remembering successful patterns, adjusting generation parameters based on past feedback, or developing user-specific models that capture particular preferences or domain knowledge. By learning from interaction history, systems could become increasingly well-aligned with individual users' needs and expectations.

Expanded Domain Coverage Our research focused on three specific domains, but the principles we've identified could potentially apply to many other domains where technical barriers limit participation. Future research could explore applying

similar approaches to areas such as data visualization, mobile app development, game design, or scientific computing. Each new domain would likely present unique challenges and opportunities, requiring adaptation of our general approaches to domain-specific constraints and user needs. However, the core principles of balancing automation with control, providing multiple expression pathways, and bridging domain knowledge with technical implementation would likely remain relevant across contexts.

Chapter 7

Conclusion

This dissertation has investigated how Artificial Intelligence can lower the technical barriers for creating and customizing interactive systems across three key domains: intelligent tutoring systems, web applications, and smart home environments. Our research introduces novel methods that bridge the gap between domain expertise and technical implementation, empowering diverse users to translate their intentions into functional systems without requiring programming knowledge.

For intelligent tutoring systems, we introduced an AI-assisted approach that empowers educators by automatically decomposing high-level teaching goals into discrete pedagogical steps. By grounding this process in established instructional strategies, our approach ensures pedagogical effectiveness while keeping educators in control of the authoring process. A preference-driven UI refinement method further allowed educators to guide the final interface design by selecting from multiple AI-generated drafts, a method that our user study with K-12 educators found enhanced both satisfaction and final interface quality compared to traditional tools.

In the domain of web applications, our contributions spanned the entire design-to-implementation workflow. We developed a transformer-based architecture for design-to-code translation that captures visual and structural relationships with high fidelity. We enhanced this approach with style awareness, allowing designers to specify aesthetics through reference images, and dynamic behavior specification through symbolic annotations in sketches. For non-technical users, we created methods for LLM-driven web development and GUI customization through natural language, enabling website creation and modification without programming expertise.

In smart home environments, we addressed the challenge of command ambiguity by introducing a multimodal disambiguation approach. When users issued subjective or underspecified commands, our system generated multiple interpretations presented as both textual descriptions and visual representations, allowing users to select the option that best matched their intent. Our user studies confirmed that this method enhances the naturalness of interaction and improves user control over system actions.

As AI continues to evolve, the potential for empowering non-technical users to create and customize technology will only grow. The principles established in this dissertation—scaffolding complex processes, mirroring expert workflows, and providing multiple pathways for expression—provide a foundation for realizing this potential. They point toward a future where technology creation is limited not by technical knowledge but is accessible to anyone with domain expertise and a creative vision.

Through continued research and development in this area, we can work toward a future where the ability to shape our digital world aligns more closely with the diversity of human needs and perspectives. The ultimate goal is to narrow the gap between having an idea and implementing it, making the tools we use true extensions of human creativity and problem-solving capacity.

References

[1] European Commission. Digital economy and society index 2021. https://ec.europa.eu/commission/presscorner/detail/en/ip_21_5481, 2021. Accessed: 2025-04-14.

[2] Christopher Scaffidi, Mary Shaw, and Brad Myers. Estimating the numbers of end users and end user programmers. In *2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05)*, pages 207–214. IEEE, 2005.

[3] Henry Lieberman, Fabio Paternò, and Volker Wulf, editors. *End User Development*. Human-Computer Interaction Series. Springer, New York, NY, USA, 2006.

[4] Advait Sarkar. Will code remain a relevant user interface for end-user programming with generative ai models? In *Proceedings of the 2023 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, Onward! 2023, page 153–167, New York, NY, USA, 2023. Association for Computing Machinery.

[5] Miriam Walker, Leila Takayama, and James A. Landay. High-fidelity or low-fidelity, paper or computer? choosing attributes when testing web prototypes. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 46(5):661–665, 2002.

[6] Sarah Suleri, Vinoth Pandian Sermuga Pandian, Svetlana Shishkovets, and Matthias Jarke. Eve: A sketch-based software prototyping workbench. In *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI EA '19, page 1–6, New York, NY, USA, 2019. Association for Computing Machinery.

[7] K.P. Moran, C. Bernal-Cárdenas, M. Curcio, R. Bonett, and D. Poshyvanyk. Machine learning-based prototyping of graphical user interfaces for mobile apps. *IEEE Trans. Softw. Eng.*, 2018.

[8] T. Beltramelli. pix2code: Generating code from a graphical user interface screenshot. In *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, page 1–6, 2018.

[9] Alex Robinson. Sketch2code: Generating a website from a paper mockup, 2019.

[10] Tuan Anh Nguyen and Christoph Csallner. Reverse engineering mobile application user interfaces with remaui. In *Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering*, ASE '15, page 248–259. IEEE Press, 2015.

[11] Z. Zhu, Z. Xue, and Z. Yuan. Automatic graphics program generation using attention-based hierarchical decoder. In *Asian Conference on Computer Vision*, pages 181–196. Springer, Cham, 2018.

[12] Sarah Suleri, Vinoth Pandian Sermuga Pandian, Svetlana Shishkovets, and Matthias Jarke. Eve: A sketch-based software prototyping workbench. In *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems (CHI EA '19)*, pages 1–6, Glasgow, Scotland, UK, 2019. Association for Computing Machinery.

[13] Fidelity or low-fidelity, paper or computer? choosing attributes when testing web prototypes, 2002.

[14] Barbara Rita Barricelli, Fabio Cassano, Daniela Fogli, and Antonio Piccinno. End-user development, end-user programming and end-user software engineering: A systematic mapping study. *Journal of Systems and Software*, 149:101–137, 2019.

[15] Jochen Rode, Mary Beth Rosson, and Manuel A. Pérez Qui~nones. End user development of web applications, 2006.

[16] Apurvanand Sahay, Arsene Indamutsa, Davide Di Ruscio, and Alfonso Pierantonio. Supporting the understanding and comparison of low-code development platforms. In *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 171–178, 2020.

[17] Katerina Tzafilkou and Nicolaos Protogeros. Diagnosing user perception and acceptance using eye tracking in web-based end-user development. *Computers in Human Behavior*, 72:23–37, 2017.

[18] Wendy E Mackay. Patterns of sharing customizable software. In *Proceedings of the 1990 ACM Conference on Computer-Supported Cooperative Work*, pages 209–221. ACM, 1991.

[19] Daniel S. Weld, Corin Anderson, Pedro Domingos, Oren Etzioni, Krzysztof Gajos, Tessa Lau, and Steve Wolfman. Automatically personalizing user interfaces. *IJCAI Proceedings*, 3:1613–1619, 2003. Available online.

[20] Leah Findlater and Joanna McGrenere. Preferences for interface adaptations: Understanding individual differences in customization behavior. *ACM Transactions on Computer-Human Interaction*, 16(1):1–44, 2009.

[21] Krzysztof Z Gajos, Katherine Everitt, Desney S Tan, Mary Czerwinski, and Daniel S Weld. Predictability and accuracy in adaptive user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1271–1274, 2008.

[22] Jamil Hussain, Anees Ul Hassan, Hafiz Syed Muhammad Bilal, Rahman Ali, Muhammad Afzal, Shujaat Hussain, Jaehun Bang, Oresti Banos, and Sungyoung Lee. Model-based adaptive user interface based on context and user experience evaluation. *Journal on Multimodal User Interfaces*, 12(1):1–16, 2018.

[23] Tae Soo Kim, DaEun Choi, Yoonseo Choi, and Juho Kim. Stylette: Styling the web with natural language. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, CHI '22, New York, NY, USA, 2022. Association for Computing Machinery.

[24] Benjamin S Bloom. The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring. *Educational Researcher*, 13(6):4–16, 1984.

[25] Wenting Ma, Olusola O Adesope, John C Nesbit, and Qing Liu. Intelligent tutoring systems and learning outcomes: A meta-analysis. *Journal of Educational Psychology*, 106(4):901, 2014.

[26] Tom Murray. An overview of intelligent tutoring system authoring tools: Updated analysis of the state of the art. *Authoring Tools for Advanced Technology Learning Environments: Toward Cost-Effective Adaptive, Interactive and Intelligent Educational Software*, pages 491–544, 2003.

[27] Jakob Nielsen. Bridging the designer-user gap. https://www.nngroup.com/articles/bridging-the-designer-user-gap/, 2023. Accessed: 2024-02-26.

[28] K.R. Koedinger, V. Aleven, N. Heffernan, B. McLaren, and M. Hockenberry. Opening the door to non-programmers: Authoring intelligent tutor behavior by demonstration. In J.C. Lester, R.M. Vicari, and F. Paraguaçu, editors, *Intelligent Tutoring Systems*, 2004.

[29] Glen Smith, Adit Gupta, and Christopher MacLellan. Apprentice tutor builder: A platform for users to create and personalize intelligent tutors, 2024.

[30] Vincent Aleven, Bruce McLaren, Jonathan Sewall, and Kenneth R Koedinger. Example-tracing tutors: A new paradigm for intelligent tutoring systems, 2009.

[31] Meghan Clark, Mark W Newman, and Prabal Dutta. Devices and data and agents, oh my: How smart home abstractions prime end-user mental models. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 1(3):1–26, 2017.

[32] Haoxiang Yu, Jie Hua, and Christine Julien. Dataset: Analysis of ifttt recipes to study how humans use internet-of-things (iot) devices. In *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*, pages 537–541, 2021.

[33] Pooja Upadhyay, Sharon Heung, Shiri Azenkot, and Robin N. Brewer. Studying exploration & long-term use of voice assistants by older adults. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, CHI '23, New York, NY, USA, 2023. Association for Computing Machinery.

[34] Alisha Pradhan, Amanda Lazar, and Leah Findlater. Use of intelligent voice assistants by older adults with low technology use. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 27(4):1–27, 2020.

[35] Ewa Luger and Abigail Sellen. "like having a really bad pa": The gulf between user expectation and experience of conversational agents. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 5286–5297, 2016.

[36] Sunyoung Kim and Abhishek Choudhury. Exploring older adults' perception and use of smart speaker-based voice assistants: A longitudinal study. *Computers in Human Behavior*, 124:106914, 2021.

[37] Benjamin R Cowan, Nadia Pantidi, David Coyle, Kellie Morrissey, Peter Clarke, Sara Al-Shehri, David Earley, and Natasha Bandeira. "what can i help you with?": Infrequent users' experiences of intelligent personal assistants. In *Proceedings of the 19th International Conference on Human-Computer Interaction with Mobile Devices and Services*, pages 1–12, 2017.

[38] Evan King, Haoxiang Yu, Sangsu Lee, and Christine Julien. Sasha: creative goal-oriented reasoning in smart homes with large language models. *arXiv preprint arXiv:2305.09802*, 2023.

[39] Dmitriy Rivkin, Francois Hogan, Amal Feriani, Abhisek Konar, Adam Sigal, Steve Liu, and Greg Dudek. Sage: Smart home agent with grounded execution. *arXiv preprint arXiv:2311.00772*, 2023.

[40] Mahda Noura, Sebastian Heil, and Martin Gaedke. Vish: Does your smart home dialogue system also need training data? In *Web Engineering: 20th International Conference, ICWE 2020, Helsinki, Finland, June 9-12, 2020, Proceedings*, volume 20, pages 171–187. Springer, 2020.

[41] R. Lutolf. Smart home concept and the integration of energy meters into a home based system. In *Seventh International Conference on Metering Apparatus and Tariffs for Electricity Supply*, pages 277–278, 1992.

[42] Charlie Wilson, Tom Hargreaves, and Richard Hauxwell-Baldwin. Smart homes and their users: A systematic analysis and key challenges. *Personal and Ubiquitous Computing*, 19:463–476, 2015.

[43] G.J. James. *The Elements of User Experience: User-Centered Design for the Web and beyond*. Pearson Education, 2 edition, 2010.

[44] R. Hartson and P.S. Pyla. *The UX Book: Process and Guidelines for Ensuring a Quality User Experience*. Morgan Kaufmann, 2 edition, 2019.

[45] T. Memmel and H. Reiterer. Support collaboration, model-based and prototyping-driven user interface specification to and creativity. *Int. J. Univ. Comput. Sci.*, 14(19):3217–3235, 2009.

[46] M. W. Newman and J. A. Landay. Sitemaps, storyboards, and specifications: A sketch of web site design practice. In *Proceedings of DIS 2000: Designing Interactive Systems*, pages 263–274, New York, New York, Aug 2000.

[47] J. I. Hong, F. C. Li, J. Lin, and J. A. Landay. End-user perceptions of formal and informal representations of web sites. In *Proceedings of Human Factors in Computing Systems: CHI 2001 Extended Abstracts*, pages 385–386, Seattle, WA, 2001. March 31–April 5, 2001.

[48] R. A. Virzi, J. L. Sokolov, and D. Karis. Usability problem identification using both low- and high-fidelity prototypes. In *Proceedings of Human Factors in Computing Systems: CHI '96*, pages 236–243, Vancouver, BC, Canada, 1996. April 13–18, 1996.

[49] T. Beltramelli. Hack your design sprint: Wireframes to prototype in under 5 minutes, 2019.

[50] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

[51] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[52] Andrés Soto, Héctor Mora, and Jaime A. Riascos. Web generator: An open-source software for synthetic web-based user interface dataset generation. *SoftwareX*, 17:100985, 2022.

[53] Vinoth Pandian Sermuga Pandian, Sarah Suleri, and Matthias Jarke. Synz: Enhanced synthetic dataset for training ui element detectors. In *26th International Conference on Intelligent User Interfaces - Companion*, IUI '21 Companion, page 67–69, New York, NY, USA, 2021. Association for Computing Machinery.

[54] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus,

S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[55] Kenton Lee, Mandar Joshi, Iulia Turc, Hexiang Hu, Fangyu Liu, Julian Eisenschlos, Urvashi Khandelwal, Peter Shaw, Ming-Wei Chang, and Kristina Toutanova. Pix2struct: Screenshot parsing as pretraining for visual language understanding, 2023.

[56] Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollar. Focal loss for dense object detection. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 42(02):318–327, feb 2020.

[57] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[58] Chunyang Chen, Ting Su, Guozhu Meng, Zhenchang Xing, and Yang Liu. From ui design image to gui skeleton: A neural machine translator to bootstrap mobile gui implementation. In *The 40th International Conference on Software Engineering, Gothenburg, Sweden*. ACM, 2018.

[59] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[60] Thiago Silva, Jean-Luc Hak, Marco Winckler, and Olivier Nicolas. A comparative study of milestones for featuring gui prototyping tools. *Journal of Software Engineering and Applications*, 10:564–589, 2017.

[61] Ali Hosseini-Khayat, Yaser Ghanam, Shelly Park, and Frank Maurer. Activestory enhanced: Low-fidelity prototyping and wizard of oz usability testing tool. In *Lecture Notes in Business Information Processing*, volume 31, pages 257–258, 2009.

[62] James A. Landay. Silk: Sketching interfaces like krazy. In *Conference Companion on Human Factors in Computing Systems (CHI '96)*, pages 398–399, Vancouver, British Columbia, Canada, 1996. Association for Computing Machinery.

[63] James Lin, Mark Newman, Jason Hong, and James Landay. Denim: An informal sketch-based tool for early stage web design, 2002.

[64] Alex Robinson. Sketch2code: Generating a website from a paper mockup, 2019. arXiv preprint `arXiv:1905.13750 [cs.CV]`.

[65] Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollar. Focal loss for dense object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(2):318–327, Feb 2020.

[66] Alex Robinson. Sketch2code: Generating a website from a paper mockup. `arXiv:1905.13750 [cs.CV]`, 2019.

[67] Chunyang Chen, Ting Su, Guozhu Meng, Zhenchang Xing, and Yang Liu. Converting ui design image to gui skeleton: A neural machine translator for bootstrapping mobile gui implementation. In *Proceedings of the 40th International Conference on Software Engineering*, Gothenburg, Sweden, 2018. ACM.

[68] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255. IEEE, 2009.

[69] Kurt Vanlehn. The behavior of tutoring systems. *Int. J. Artif. Intell. Ed.*, 16(3):227–265, aug 2006.

[70] Kurt VanLehn. The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems. *Educational Psychologist*, 46(4):197–221, 2011.

[71] Benedict Du Boulay. Artificial intelligence as an effective classroom assistant. *IEEE Intelligent Systems*, 31(6):76–81, 2016.

[72] Saiying Steenbergen-Hu and Harris Cooper. A meta-analysis of the effectiveness of intelligent tutoring systems on college students' academic learning. *Journal of Educational Psychology*, 106(2):331–347, 2014.

[73] A. Mitrovic, B. Martin, P. Suraweera, K. Zakharov, N. Milik, J. Holland, and N. McGuigan. Aspire: an authoring system and deployment environment for constraint-based tutors. *International Journal of Artificial Intelligence in Education*, 19(2):155–188, 2009.

[74] Noboru Matsuda, William W Cohen, Jonathan Sewall, and Kenneth R Koedinger. Simstudent: a machine learning agent that models student learning for tutor authoring. In *Proceedings of the 2007 conference on Artificial Intelligence in Education: Building Technology Rich Learning Contexts That Work*, pages 599–601. IOS Press, 2007.

[75] Christopher J MacLellan, Kenneth R Koedinger, and Noboru Matsuda. Authoring tutors with simstudent: An evaluation of efficiency and model quality. In *Intelligent Tutoring Systems: 12th International Conference, ITS 2014, Honolulu, HI, USA, June 5-9, 2014. Proceedings 12*, pages 551–560. Springer, 2014.

[76] C.J. MacLellan and K.R. Koedinger. Domain-general tutor authoring with apprentice learner models. *Int J Artif Intell Educ*, 32(1):76–117, 2022.

[77] Tommaso Calo and Christopher Maclellan. Towards educator-driven tutor authoring: Generative ai approaches for creating intelligent tutor interfaces. In *Proceedings of the Eleventh ACM Conference on Learning @ Scale*, L@S '24, page 305–309, New York, NY, USA, 2024. Association for Computing Machinery.

[78] J.D. Zamfirescu-Pereira, Richmond Y. Wong, Bjoern Hartmann, and Qian Yang. Why johnny can't prompt: How non-ai experts try (and fail) to design llm prompts. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, CHI '23, New York, NY, USA, 2023. Association for Computing Machinery.

[79] Bryan Wang, Gang Li, and Yang Li. Enabling conversational interaction with mobile ui using large language models. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, pages Article 432, 1–17, 2023.

[80] Savvas Petridis, Michael Terry, and Carrie Jun Cai. Promptinfuser: Bringing user interface mock-ups to life with large language models. In *Extended Abstracts of the 2023 CHI Conference on Human Factors in Computing Systems*, pages Article 237, 1–6, 2023.

[81] Jakob Nielsen. Generative ui and outcome-oriented design, 2023.

[82] Ben Shneiderman. *Human-centered AI*. Oxford University Press, 2022.

[83] Enkelejda Kasneci, Kathrin Seßler, Stefan Küchemann, Maria Bannert, Daryna Dementieva, Frank Fischer, Urs Gasser, Georg Groh, Stephan Günnemann, Eyke Hüllermeier, et al. Chatgpt for good? on opportunities and challenges of large language models for education. *Learning and individual differences*, 103:102274, 2023.

[84] Qingyao Li, Lingyue Fu, Weiming Zhang, Xianyu Chen, Jingwei Yu, Wei Xia, Weinan Zhang, Ruiming Tang, and Yong Yu. Adapting large language models for education: Foundational capabilities, potentials, and challenges. *arXiv preprint arXiv:2401.08664*, 2023.

[85] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[86] Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*, 2022.

[87] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022.

[88] Yiran Wu, Feiran Jia, Shaokun Zhang, Qingyun Wu, Hangyu Li, Erkang Zhu, Yue Wang, Yin Tat Lee, Richard Peng, and Chi Wang. An empirical study on challenging math problem solving with gpt-4. *arXiv preprint arXiv:2306.01337*, 2023.

[89] Xiaowu Zhang, Xiaotian Zhang, Cheng Yang, Hang Yan, and Xipeng Qiu. Does correction remain an problem for large language models? *arXiv preprint arXiv:2308.01776*, 2023.

[90] Jialu Zhang, José Cambronero, Sumit Gulwani, Vu Le, Ruzica Piskac, Gustavo Soares, and Gust Verbruggen. Repairing bugs in python assignments using large language models. *arXiv preprint arXiv:2209.14876*, 2022.

[91] Tung Do Viet and Konstantin Markov. Using large language models for bug localization and fixing. In *2023 12th International Conference on Awareness Science and Technology (iCAST)*, pages 192–197. IEEE, 2023.

[92] Lin Ni, Sijie Wang, Zeyu Zhang, Xiaoxuan Li, Xianda Zheng, Paul Denny, and Jiamou Liu. Enhancing student performance prediction on learnersourced questions with sgnn-llm synergy. *arXiv preprint arXiv:2309.13500*, 2023.

[93] Bor-Chen Kuo, Frederic TY Chang, and Zong-En Bai. Leveraging llms for adaptive testing and learning in taiwan adaptive learning platform (talp), 2023.

[94] Unggi Lee, Sungjun Yoon, Joon Seo Yun, Kyoungsoo Park, YoungHoon Jung, Damji Stratton, and Hyeoncheol Kim. Difficulty-focused contrastive learning for knowledge tracing with a large language model-based difficulty prediction. *arXiv preprint arXiv:2312.11890*, 2023.

[95] Gautam Yadav, Ying-Jui Tseng, and Xiaolin Ni. Contextualizing problems to student interests at scale in intelligent tutoring system using large language models. *arXiv preprint arXiv:2306.00190*, 2023.

[96] Marianna Nezhurina, Lucia Cipolina-Kun, Mehdi Cherti, and Jenia Jitsev. Alice in wonderland: Simple tasks showing complete reasoning breakdown in state-of-the-art large language models. *arXiv preprint arXiv:2406.02061*, 2024.

[97] Ehsan Latif, Gengchen Mai, Matthew Nyaaba, Xuansheng Wu, Ninghao Liu, Guoyu Lu, Sheng Li, Tianming Liu, and Xiaoming Zhai. Artificial general intelligence (agi) for education. *arXiv preprint arXiv:2304.12479*, 2023.

[98] Lixiang Yan, Lele Sha, Linxuan Zhao, Yuheng Li, Roberto Martinez-Maldonado, Guanliang Chen, Xinyu Li, Yueqiao Jin, and Dragan Gašević. Practical and ethical challenges of large language models in education: A systematic scoping review. *British Journal of Educational Technology*, 55(1):90–112, 2024.

[99] Silvia Milano, Joshua A McGrane, and Sabina Leonelli. Large language models challenge the future of higher education. *Nature Machine Intelligence*, 5(4):333–334, 2023.

[100] Lars Krupp, Steffen Steinert, Maximilian Kiefer-Emmanouilidis, Karina E Avila, Paul Lukowicz, Jochen Kuhn, Stefan Küchemann, and Jakob Karolus. Challenges and opportunities of moderating usage of large language models in education. *arXiv preprint arXiv:2312.14969*, 2023.

[101] Damien Masson, Sylvain Malacria, Géry Casiez, and Daniel Vogel. Directgpt: A direct manipulation interface to interact with large language models. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, CHI '24, New York, NY, USA, 2024. Association for Computing Machinery.

[102] Zekun Zeng, Xiaohua Sun, and Xin Liao. Artificial intelligence augments design creativity: A typeface family design experiment. In Aaron Marcus and Wentao Wang, editors, *Design, User Experience, and Usability. User Experience in Advanced Technological Environments*, pages 400–411, Cham, 2019. Springer International Publishing.

[103] Stuart K. Card, Thomas P. Moran, and Allen Newell. The keystroke-level model for user performance time with interactive systems. *Commun. ACM*, 23(7):396–410, jul 1980.

[104] Nick Symmonds. Visual web developer, 01 2006.

[105] Nishant Sinha, Rezwana Karim, and Monika Gupta. Simplifying web programming. In *Proceedings of the 8th India Software Engineering Conference*, ISEC '15, page 80–89, New York, NY, USA, 2015. Association for Computing Machinery.

[106] Giuseppe Ghiani, Fabio Paternò, Lucio Davide Spano, and Giuliano Pintori. An environment for end-user development of web mashups. *International Journal of Human-Computer Studies*, 87:38–64, 2016.

[107] Robert Waszkowski. Low-code platform for automating business processes in manufacturing. *IFAC-PapersOnLine*, 52(10):376–381, 2019. 13th IFAC Workshop on Intelligent Manufacturing Systems IMS 2019.

[108] Davide Di Ruscio, Dimitris Kolovos, Juan Lara, Alfonso Pierantonio, Massimo Tisi, and Manuel Wimmer. Low-code development and model-driven engineering: Two sides of the same coin? *Software and Systems Modeling*, 21, 01 2022.

[109] Pedro M. Gomes and Miguel A. Brito. Low-code development platforms: A descriptive study. In *2022 17th Iberian Conference on Information Systems and Technologies (CISTI)*, pages 1–4, 2022.

[110] Marcus Woo. The rise of no/low code software development—no experience needed? *Engineering*, 6, 07 2020.

[111] Md Abdullah Al Alamin, Sanjay Malakar, Gias Uddin, Sadia Afroz, Tameem Haider, and Anindya Iqbal. An empirical study of developer discussions on low-code software development challenges, 05 2021.

[112] Sebastian Käss, Susanne Strahringer, and Markus Westner. Drivers and inhibitors of low code development platform adoption. In *2022 IEEE 24th Conference on Business Informatics (CBI)*, volume 01, pages 196–205, 2022.

[113] Yajing Luo, Peng Liang, Chong Wang, Mojtaba Shahin, and Jing Zhan. Characteristics and challenges of low-code development: The practitioners' perspective. In *Proceedings of the 15th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, ESEM '21, New York, NY, USA, 2021. Association for Computing Machinery.

[114] Tom Page. Personalisation of web interfaces: A study of user preferences and implications for user interface design. *Journal of Design Research*, 14(1):22–53, 2016.

[115] Krzysztof Z Gajos, Jacob O Wobbrock, and Daniel S Weld. Automatically generating personalized user interfaces with supple. In *Proceedings of the 9th international conference on Intelligent user interfaces*, pages 93–100, 2007.

[116] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

[117] Chenfei Wu, Shengming Yin, Weizhen Qi, Xiaodong Wang, Zecheng Tang, and Nan Duan. Visual chatgpt: Talking, drawing and editing with visual foundation models, 2023.

[118] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[119] Forrest Huang, Gang Li, Xin Zhou, John F Canny, and Yang Li. Creating user interface mock-ups from high-level text descriptions with deep-learning models. *arXiv preprint arXiv:2110.07775*, 2021.

[120] 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, New York, NY, USA, 2018. Association for Computing Machinery.

[121] Tommaso Calò and Luigi De Russis. Style-aware sketch-to-code conversion for the web. In *Companion of the 2022 ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS '22 Companion, page 44–47, New York, NY, USA, 2022. Association for Computing Machinery.

[122] OpenAI. Gpt-4 technical report, 2023.

[123] Jules White, Quchen Fu, Sam Hays, Michael Sandborn, Carlos Olea, and Henry Gilbert. A prompt pattern catalog to enhance prompt engineering with chatgpt. *arXiv preprint arXiv:2302.11382*, 2023.

[124] Mirac Suzgun, Mitchell Scales, Jing Ni, Eric Wang, Ben McArthur, Mohit Shridhar, Nate Schärli, and John Schulman. Promptbreeder: Self-referential self-improvement via prompt evolution. *arXiv preprint arXiv:2309.16797*, 2023.

[125] Qiaoyu Liu, Xin Chen, and Haoming Deng. A systematic survey of prompt engineering. *arXiv preprint arXiv:2307.05242*, 2023.

[126] Jason Wu, Siyan Wang, Siman Shen, Yi-Hao Peng, Jeffrey Nichols, and Jeffrey P Bigham. Webui: A dataset for enhancing visual ui understanding with web semantics. *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, pages 1–14, 2023.

[127] John Brooke. Sus: A 'quick and dirty'usability scale. *Usability evaluation in industry*, 189(194):4–7, 1996.

[128] Rob Dunne, Tim Morris, and Simon Harper. A survey of ambient intelligence. *ACM Computing Surveys (CSUR)*, 54(4):1–27, 2021.

[129] Mark Weiser. The computer for the 21st century. *ACM SIGMOBILE Mobile Computing and Communications Review*, 3(3):3–11, 1999.

[130] Dongsu Kim, Yeobeom Yoon, Jongman Lee, Pedro J Mago, Kwangho Lee, and Heejin Cho. Design and implementation of smart buildings: A review of current research trend. *Energies*, 15(12):4278, 2022.

[131] Artur Branny, Maja Steen Møller, Silviya Korpilo, Timon McPhearson, Natalie Gulsrud, Anton Stahl Olafsson, Christopher M Raymond, and Erik Andersson. Smarter greener cities through a social-ecological-technological systems approach. *Current Opinion in Environmental Sustainability*, 55:101168, 2022.

[132] Giovanni Acampora, Diane J Cook, Parisa Rashidi, and Athanasios V Vasilakos. A survey on ambient intelligence in healthcare. *Proc. IEEE*, 101(12):2470–2494, 2013.

[133] Can an intelligent personal assistant (ipa) be your friend? para-friendship development mechanism between ipas and their users, 2020.

[134] Michaela R Reisinger, Sebastian Prost, Johann Schrammel, and Peter Fröhlich. User requirements for the design of smart homes: Dimensions and goals. *Journal of Ambient Intelligence and Humanized Computing*, pages 1–20, 2022.

Appendix A

Publications

Publications updated as of June 2025.

A.1 International Journals

1. Tommaso Calò and Luigi De Russis (2025) **Advancing Code Generation from Visual Designs through Transformer-Based Architectures and Specialized Datasets** in: Proceedings of the ACM on Human-Computer Interaction (PACMHCI), ACM, Volume 9, in press.
   DOI: 10.1145/3734190

2. Tommaso Calò and Luigi De Russis (2024) **Enhancing smart home interaction through multimodal command disambiguation** in: Personal and Ubiquitous Computing, Springer, pages: 1–16,
   DOI: 10.1007/s00779-024-01827-3

A.2 Proceedings

1. Tommaso Calo and Christopher Maclellan (2024) **Towards Educator-Driven Tutor Authoring: Generative AI Approaches for Creating Intelligent Tutor Interfaces** in: Proceedings of the Eleventh ACM Conference on Learning @ Scale, Association for Computing Machinery, Atlanta, GA, USA, pages: 305–309,

ISBN: 9798400706332
DOI: 10.1145/3657604.3664694

2. Tommaso Calò and Luigi De Russis (2023) **Leveraging Large Language Models for End-User Website Generation** in: End-User Development: 9th International Symposium, IS-EUD 2023, Cagliari, Italy, June 6–8, 2023, Proceedings, Springer-Verlag, Berlin, Heidelberg, pages: 52–61,
ISBN: 978-3-031-34432-9
DOI: 10.1007/978-3-031-34433-6_4

3. Tommaso Calò and Luigi De Russis (2022) **Style-Aware Sketch-to-Code Conversion for the Web** in: EICS '22 Companion, Association for Computing Machinery, New York, NY, USA, Sophia Antipolis, France, pages: 44–47,
ISBN: 9781450390316
DOI: 10.1145/3531706.3536462

4. Tommaso Calò and Luigi De Russis (2022) **Creating Dynamic Prototypes from Web Page Sketches** in: Proceedings of the 1st ACM SIGPLAN International Workshop on Programming Abstractions and Interactive Notations, Tools, and Environments, Association for Computing Machinery, New York, NY, USA, Auckland, New Zealand, pages: 20–25,
ISBN: 9781450399104
DOI: 10.1145/3563836.3568724

A.3 Other Publications

1. Tommaso Calo' and Luigi De Russis (2025) **MorphGUI: Real-time Natural Language Interface Customization with Large Language Models**, Under review at International Journal of Human-Computer Studies

2. Tommaso Calo' and Christopher Maclellan (2025) **AI-Assisted Interface Design for Intelligent Tutoring Systems**, Under submission at CHI 2026