Effective GUI Generation: Leveraging Large Language Models for Automated GUI Prototyping

Lennart Fiebig[1], Kristian Kolthoff[1], Christian Bartelt[1], Simone Paolo Ponzetto[2]
[1]Institute for Enterprise Systems (InES), University of Mannheim
[2]Data and Web Science Group, University of Mannheim
fiebiglennart@gmail.com, kolthoff@es.uni-mannheim.de, bartelt@uni-mannheim.de, simone@informatik.uni-mannheim.de

Abstract

GUI prototyping is a common technique for requirements elicitation during software development and is essential in visualizing and communicating user requirements. However, creating GUI prototypes can be resource-intensive in terms of time and cost. This paper explores the innovative approach of integrating Large Language Models (LLMs) into the development of Graphical User Interface (GUI) prototypes to address this challenge. The primary objective of this work is to convert high-level text descriptions of mobile app interfaces into precise and detailed GUI prototypes. Various prompting strategies are adapted and evaluated within a structured GUI prototyping framework. The findings of this work highlight the feasibility of combining a state-of-the-art LLM with structured prompting approaches. This combination has successfully created high-quality GUI prototypes from textual descriptions, showcasing the significant potential of LLMs in the realm of GUI prototyping.

1. Introduction

Requirements elicitation (RE) is a crucial step in software engineering that involves collecting and interpreting user needs to define the functionality of a system. RE is a critical phase where misunderstanding or misinterpretations can lead to project delays, increased costs, and products that fail to meet user expectations ("Requirements engineering processes, tools/technologies, & methodologies", 2009). Prior research proposed a variety of techniques to facilitate and execute RE effectively ("Requirements engineering processes, tools/technologies, & methodologies", 2009; Tiwari, Rathore, & Gupta, 2012). Some of the most popular techniques are interviews, which are direct conversations with stakeholders to gather in-depth and precise information. Moreover, GUI prototyping also plays a significant role for effective RE. The created GUI prototypes are often employed with other RE techniques like interviews and used to represent the requirements visually. This visual approach is instrumental in clarifying user needs, as relying solely on natural language can result in ambiguous or incomplete requirements (Aldave, Vara, Granada, & Marcos, 2019). However, despite the valuable insights gained, the investment in time and cost for developing these GUI prototypes can be significant ("Requirements engineering processes, tools/technologies, & methodologies", 2009) and therefore poses a challenge.

The emergence of Large Language Models (LLMs) offers a promising opportunity to facilitate this aspect of software development. With their advanced natural language understanding and generative capabilities, LLMs have the potential to automate GUI prototyping. They could drastically reduce time and costs by interpreting textual requirements and automatically generating corresponding GUIs. This paper explores the practicality of using LLMs to create GUI prototypes, focusing on functional requirements that can potentially be used as initial starting points in the RE process. This is achieved by generating HTML and CSS code from high-level text descriptions for GUI prototypes. In addition to examining the capability of state-of-the-art LLMs, the study aims to assess the influence of various prompting techniques. This includes assessing the effectiveness of guiding the LLM through a step-by-step GUI prototyping process, which is closer to a human expert approach, and exploring in-context learning (Dong et al., 2022). We publicly provide the source code, prompts and evaluation data in our accompanying repository (*Repository*, 2024).

2. Background

The term *Large Language Model (LLM)* refers to Transformer-based models with hundreds of billions of parameters trained on massive amounts of textual data (Shanahan, 2024). The first model, for which the term LLM was coined is *GPT-3* (Brown et al., 2020). *GPT-3*

URI: https://hdl.handle.net/10125/109684
978-0-9981331-8-8

is built on the architecture of its predecessor, *GPT-2*, (Radford et al., 2019) but vastly larger, with an increase from 1.5 billion to 175 billion parameters. The model presents a groundbreaking milestone in the evolution of language models, showing that scaling up the size of neural networks leads to a considerable enhancement in capabilities. *GPT-3* excelled in many NLP tasks without task-specific fine-tuning, nearing the performance of fine-tuned systems in some cases (Brown et al., 2020).

LLMs, including *GPT-3*, exhibit unique emergent abilities, such as in-context learning, enabling the model to understand and perform tasks based on instructions or examples without specific training (Dong et al., 2022). The precise cause behind these emergent abilities remains unclear, raising the question of whether further scaling of LLMs might reveal more of such capabilities (Wei, Tay, et al., 2022).

GPT-4 (Achiam et al., 2023), the newest model in the GPT series from OpenAI, stands out as a multi-modal model capable of processing both text and image inputs. While specific details like parameter count and training data volume have not been publicly disclosed, it is reported that both are significantly larger than those of previous GPT models. *GPT-4* has demonstrated near-human performance levels across various professional and academic benchmarks, further pushing the boundaries of text understanding and reasoning capabilities (Achiam et al., 2023).

3. Related Work

GUI Prototyping tools such as *Adobe XD* (*Adobe XD*, 2024) or *Figma* (*Figma*, 2024) are widely used for designing prototypes of websites, mobile apps and other digital products. They provide libraries of diverse design and UI components, along with a variety of templates and layouts to enable rapid creation of GUI prototypes within a visual editor. However, creating prototypes with such tools can be time-consuming, and their complexity necessitates a certain level of design expertise from users to effectively utilize their full potential. Hence, numerous strategies have been introduced to streamline GUI prototyping. In the early days of software development, sketch-based prototyping was a common practice. It offered a straightforward and cost-effective way to create GUI prototypes using pen and paper. However, the transition from paper sketches to software often encountered errors due to the subjective nature of sketches (Rocha, Hak, Winckler, & Nicolas, 2019). To address this, approaches such as *sketchiXML* (Coyette, Faulkner, Kolp, Limbourg, & Vanderdonckt, 2004) and *Draw2Code* (Sonje, Dave, Pardeshi, & Chaudhari, 2022) were developed to detect UI components from sketches and convert them into code, enhancing sketch-based prototyping. Beyond sketch-based inputs, several methods generate code from existing prototypes and GUI screenshots. *Pix2code* (Beltramelli, 2018) was the first to reverse engineer GUIs from screenshots using machine learning. These tools effectively convert existing ideas represented as sketches or screenshots into prototypes that can be further customized. Nevertheless, they lack the facilitation of the creative aspect of uncovering new or unexplored requirements. Consequently, they do not significantly contribute innovative ideas to requirements elicitation. Besides approaches that create GUI prototypes, various methods utilize existing GUIs or sketches to retrieve GUI designs aligned with or related to a given input. For example, approaches like *GUIFetch* (Behrang, Reiss, & Orso, 2018) and *Swire* (Z. Huang, 2020) exploit large-scale GUI repositories collected from scraped apps to retrieve GUIs similar to a provided sketch. Similarly, retrieval approaches like *Screen2Vec* (Li, Popowski, Mitchell, & Myers, 2021), and *VINS* (Bunian et al., 2021) fetch existing GUIs based on images or screenshots of current prototypes. While these approaches are primarily focused on providing multiple design choices or drawing inspiration from existing GUIs, they are less useful for facilitating RE.

All approaches mentioned previously require visual inputs, however, there has been an increasing interest in using natural language for GUI prototyping, although this area remains relatively unexplored. For instance, one of these early approaches is *RaWi* (Kolthoff, Bartelt, & Ponzetto, 2023, 2020), a prototyping approach that retrieves existing GUIs from a large-scale GUI repository employing textual input. A comparable approach to our work is from Huang et al. (F. Huang, Li, Zhou, Canny, & Li, 2021). They designed a generative deep neural network to create GUI prototypes from high-level text descriptions. However, the generated prototypes are low-fidelity mockups that merely provide first ideas for design and layout. In comparison, in our approach the LLM generates GUI prototypes with high fidelity, encompassing many GUI functionalities and proposing concrete GUI feature implementations.

4. Methodology

While LLMs are primarily designed to predict the next word in a sequence, their application extends beyond text-based tasks. These models have been effectively utilized in other domains such as protein sequences, audio processing, and software code generation (Eloundou, Manning, Mishkin, & Rock,

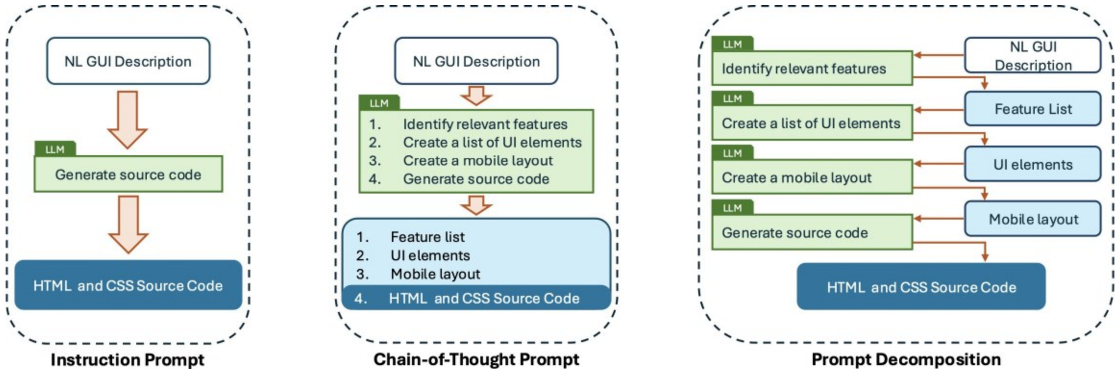

Figure 1. Illustration of the three prompting approaches utilized in our evaluation: Instruction Prompting, Chain-of-Thought Prompting (ZS and FS variants), and Prompt Decomposition (ZS and FS variants).

2023). Notably, their use in code creation has demonstrated impressive results (Achiam et al., 2023; Chen et al., 2021).

Therefore, we focus on generating GUI prototypes in HTML and CSS in our work. This approach is highly advantageous for eliciting requirements, as the resulting prototypes can be directly used as an initial starting point. Additionally, HTML and CSS code can be easily imported into well-known prototyping tools such as *Figma* (*Figma*, 2024) to adapt them in the process.

This study evaluates various prompt engineering approaches to identify best practices for enhancing the LLMs GUI generation capabilities. A prompt is the textual instruction for the LLM and plays a pivotal role in shaping the response of the model. This input directs the understanding of the model and impacts the nature of its output (Liu et al., 2023). A prompt might range from a straightforward question or a problem statement to a complex set of instructions. The ability of an LLM to generate helpful and contextually relevant responses heavily relies on the precision and quality of these prompts. Therefore, this research aims to identify optimal strategies for prompt construction that enhance the quality of the created GUI prototypes in the form of HTML and CSS code. We constructed five distinct prompts utilizing different prompt engineering concepts, *(i)* an *Instruction* prompt, *(ii)* a *Chain-of-Thought* prompt *(CoT-ZS)*, *(iii)* a *Chain-of-Thought* prompt enhanced with few-shot examples *(CoT-FS)*, *(iv) Prompt Decomposition (PD-ZS)* and *(v) Prompt Decomposition* enhanced with few-shot examples *(PD-FS)*. All prompts were evaluated on three different metrics, using a dataset of brief high-level text descriptions of real-world mobile GUIs. Subsequently, we present the employed prompts.

4.1. Instruction Prompting

The simplest method to prompt an LLM involves describing the task to the model without offering a few examples or additional guidance. This prompt solely consists of the task description of generating the HTML and CSS code for a respective mobile page description and is named as the *Instruction* prompt.

4.2. Chain-of-Thought Prompting

As an enhancement for the *Instruction* prompt, we utilized the concept of *Chain-of-Thought (CoT)* (Wei, Wang, et al., 2022) to emulate a more human-like approach to the task. Typically, a person tasked with creating GUI prototypes would not start with the prototype implementation directly but rather deconstruct the problem into smaller, more manageable steps. *CoT* prompting grants LLMs similar capabilities, prompting them to articulate a detailed, step-by-step breakdown of their reasoning process. The most straightforward utilization of *CoT* is *Zero-Shot (ZS) CoT* prompting. In this method, the model is instructed to reason out its response. This can be achieved by incorporating a phrase like ”Let’s think step by step”, as initially suggested by Kojima et al. (Kojima, Gu, Reid, Matsuo, & Iwasawa, 2022), into the prompt. However, initial tests with the standard *CoT-ZS* approach revealed that the model primarily focused on reasoning out the creation of the source code rather than conceptualizing and designing the GUI prototype, leading to rudimentary results. To address this, rather than allowing the model to formulate a reasoning process independently, it was instead guided by a structured four-step GUI prototyping framework:

1. Create a list of potentially required or relevant GUI features for the described GUI.
2. Create a list of GUI components required to display the GUI features identified in the first step.
3. Create a cohesive mobile layout for GUI comps.
4. Generate the HTML and CSS code for a mobile page based on the previously generated contents.

Integrating this reasoning sequence into the prompt aimed to direct the output of the LLMs more effectively. The objective was to ensure that the model generated the required code after engaging in the analytical and design processes necessary to create a well-thought-out GUI.

4.3. Prompt Decomposition

The *CoT* prompt encapsulates the entire GUI prototyping process in a single comprehensive generation, resulting in complex and extensive prompts and outputs. To possibly enhance the effectiveness of the *CoT* prompt, we utilized a concept known as *Prompt Decomposition (PD)* (Wang, Deng, & Sun, 2022; Zhou et al., 2022). This approach focuses on dividing the prompt of the model into multiple cognitive stages by simplifying complex tasks into smaller, more manageable sub-tasks, each addressed with a separate prompt. This allows the LLM to provide more computation capacity for each step. For our *PD* prompt, we split the *CoT* prompt into four steps, so that each step in the GUI prototyping framework has its own prompt. The output from each prompt serves as the contextual foundation for the following one, as shown in Fig. 1. This technique lets the model focus on each step individually, simplifying the prompts the model has to process and the generated output simultaneously.

4.4. Few-Shot Prompting

We utilized *Few-Shot (FS)* prompting for both structured approaches to further improve the *CoT-ZS* and *PD-ZS* prompts by providing more specific guidance and real-world reference points. Few-shot prompting, proposed in the foundational paper of *GPT-3* (Brown et al., 2020), involves providing a couple of input-output examples before requesting the model to apply the task to a new, unseen input. This approach depends on the ability of the model to learn from these limited examples, enabling it to handle tasks it has not seen before, a process known as in-context learning (Dong et al., 2022).

In particular, we developed two-shot prompts, enhancing the *CoT-ZS* prompt and the *PD-ZS* approach with two distinct input-output pairs. These examples drew inspiration from two different GUIs in the *Rico* dataset, which is a large-scale dataset of real-world mobile GUIs (Deka et al., 2017). Both GUIs were manually reconstructed, following our four-step prototyping framework. This led to the HTML code for the reconstructed GUIs mirroring the original *Rico* examples and the intermediate results for the conceptualization, including the list of GUI features, the GUI components, and the corresponding GUI layout. The intermediate results and the final GUI prototypes were instrumental in directing the LLM responses best.

5. Experimental Evaluation

Subsequently, we present our three research questions guiding the experimental evaluation. Additionally, we explain the detailed evaluation setup and methodology for assessing the effectiveness of using state-of-the-art LLMs combined with various prompting techniques for GUI prototype generation.

5.1. Research Questions

This research uses an LLM to create GUI prototypes based on high-level text descriptions. In this realm, the objective is to answer the following research questions:

- **RQ1:** Can structured prompting improve the LLMs effectiveness of generating GUI prototypes over unstructured prompting?
- **RQ2:** Can Prompt Decomposition improve the LLMs effectiveness of generating GUI prototypes over Chain-of-Thought prompting?
- **RQ3:** Can Few-shot prompting improve the LLMs effectiveness of generating GUI prototypes over Zero-shot prompting?

To answer these questions, we gathered a dataset of textual mobile GUI descriptions and employed it to generate GUI prototypes by leveraging a state-of-the-art LLM in combination with various prompts. We then evaluated these prototypes using three distinct metrics to gain insights into the performance of established prompting methods and determine their effectiveness in enhancing LLMs for GUI prototyping. The data from this evaluation was analyzed to obtain research findings.

5.2. Dataset

To ensure that the generated prototypes were grounded in reality and practicality, a dataset encompassing brief text descriptions of actual GUIs was essential. During the initial phase of this research, we utilized *Screen2Words* (Wang et al., 2021). This unique dataset comprises text descriptions for more than 20,000 distinct *Rico* screens, each annotated by five human annotators. However, the dataset showed significant variability in the accuracy of its descriptions. Additionally, it contains numerous contradictory and incorrect entries, as well as overly high-level and ambiguous descriptions (e.g., *"page of an e-commerce website"*). For these reasons, we decided not to use this dataset for our evaluation and created a new dataset.

Instead of utilizing *Screen2Words*, we took an innovative approach by creating a unique dataset of GUI descriptions using *GPT-4*. *GPT-4* is a multi-modal LLM capable of processing images and text inputs (Achiam et al., 2023). This ability makes it an ideal candidate for generating concise text descriptions from GUI screenshots. Therefore, we were able to create a dataset with high-level GUI text descriptions, each uniformly providing accurate representations of a corresponding *Rico* GUI. Related studies demonstrated the ability of *GPT-4* for screen interpretation and summarization (Wang, Li, & Li, 2023; Kolthoff, Kretzer, Bartelt, Maedche, & Ponzetto, 2024). In their study, Wang et al. (Wang et al., 2023) assessed the predecessor of *GPT-4*, *GPT-3*, in creating brief descriptions of mobile GUIs from HTML structures. These descriptions were benchmarked against *Screen2Words* (Wang et al., 2021) for their accuracy in depicting *Rico* GUIs, where *GPT-3* demonstrated a superior performance. Additionally, Yan et al. (Yan et al., 2023) achieved remarkable results in evaluating *GPT-4* on navigating smartphone screens and determining subsequent actions to fulfill given instructions. These results illustrate the capabilities of *GPT-4* in action reasoning, precise localization, and, crucially, screen interpretation based on images. These studies evidence the capabilities of *GPT-4*, which reinforces its suitability for creating accurate textual descriptions from GUI screenshots.

Rico (Deka et al., 2017), a large-scale GUI dataset previously utilized in the training and evaluation of other GUI prototyping methods (Kolthoff et al., 2023; F. Huang et al., 2021) provided the basis for these text descriptions. As currently the largest publicly available dataset for mobile GUIs, *Rico* includes more than 72,000 unique GUIs from over 9,700 Android applications across 27 app categories. For the creation of our dataset, 30 different GUIs from the *Rico* dataset were sampled. The selection process was guided by a series of specific criteria each GUI needed to possess. First, any game interfaces were excluded, as the focus was on more practical and everyday applications. Also, loading screens and GUIs dominated by large images were eliminated, as they often lacked a detailed functional complexity. By selecting GUIs with complex functionality, we aimed to increase the challenge for the LLM to generate an accurate prototype.

Generic interfaces like login screens or pop-ups were also dismissed. The rationale was that such standard GUIs are not very important in the requirements elicitation scenario. Furthermore, the selection aimed to encompass a broad spectrum of domains and functionalities. As a result, our dataset spans across 21 unique app domains, with each GUI being distinct regarding its main functionality. By focusing on this varied collection of domains and functionalities, the dataset is designed to provide a robust assessment of the LLMs GUI generation abilities. Following the selection of the *Rico* GUIs, the next step involved was generating concise text descriptions for each GUI.

We created a two-shot prompt to ensure that each description contained the same information, ensuring consistency in the synthesized data. First, we extracted the category or genre of the app from which the page is derived (e.g., *shopping app*, *music app*). The intention behind including this information is to utilize the extensive domain knowledge of the LLM for the GUI prototyping process. The second bit of information was a brief description of the primary purpose or critical function of the page to create. This part of the description aims to capture the main action or feature of the page, such as displaying products for sale or presenting information about a restaurant. The intent is to keep this description concise, in order to challenge the LLM to develop features and functionalities that fulfill the stated intent independently. We manually verified the resulting text descriptions to ensure the correctness of the generated responses. Here is an example of the generated GUI descriptions: *"A page from a weather forecasting app, presenting current and upcoming weather conditions"*.

5.3. Large Language Model

To conduct the experiments, we employed OpenAI's *GPT-4* model to execute the constructed prompts. We chose *GPT-4* because, at the time of this research, it was the top-performing LLM across various benchmarks, including code generation (Achiam et al., 2023). Specifically, we used the *gpt-4-1106-preview* model (accessed in January 2024) (*gpt-4-1106-preview*, 2024), the most recent release during the study. A consistent temperature setting of 0.7 was maintained for the model across all experiments, providing a reasonable trade-off between reproducibility and creativity for prototyping.

5.4. Evaluation Methodology

This chapter details the methodology used for evaluating the GUI prototypes generated for the brief textual descriptions. The goal is to assess the LLM's performance in this task and identify differences across the prompting approaches. These insights are obtained by evaluating each generated GUI prototype, focusing on three essential metrics: *(i)* accuracy of GUI features, *(ii)* quantity of GUI features and *(iii)* visual appeal. These metrics help determine the LLM's effectiveness in creating useful GUI prototypes that encompass many functionalities, align with the text descriptions and thus provide good initial starting points for elicitation.

For conducting the evaluation, we recruited 20 expert crowd-workers from the crowdsourcing platform *Prolific* (Palan & Schitter, 2018), which has been shown to provide higher data quality compared to other popular crowdsourcing platforms (Douglas, Ewell, & Brauer, 2023). In order to additionally increase the data quality, we restricted the annotation to crowd-workers that were fluent in English, had an approval rate of >99% and at least 30 prior submissions. Participants (Male:80%, Female:15%, Diverse:5%) had an age of 30.8 (σ=8.15) on average and had a self-reported knowledge of responsive design, UI design and UX with an average of 3.6 (σ=2.84) years of experience in creating GUI prototypes and an average of 3.3 (σ=2.45) years of experience in evaluating GUI prototypes, respectively. Moreover, we included several attention checks that participants were required to pass. For each of the 30 diverse GUI text descriptions, we generated five GUIs employing the described techniques (*Instruction*, *CoT-ZS*, *CoT-FS*, *PD-ZS* and *PD-FS*) resulting in a total of 150 generated GUIs. Each of them was annotated by ten different crowd-workers across three different metrics, resulting in a comprehensive total of 4,500 collected annotations in our evaluation.

Participants were presented with all five GUI prototypes (as screenshots) generated from the same textual description simultaneously, to enable a comparative assessment of the prompt engineering techniques. To minimize bias, the origins of the prototypes, in terms of which prompt produced which prototype, were not disclosed to the evaluators and the ordering was randomized. Evaluators were assigned to evaluate each prototype on all three metrics separately, using a Likert scale from 1 (*very low*) to 5 (*very high*).

5.5. Evaluation Metrics

Before the descriptions and the GUI screenshots were provided to participants for annotation, each participant was given a brief textual introduction to the annotation task and introduced to the role of GUI prototypes within the RE process. In addition, instructions and examples on assessing each evaluation metric were provided to participants. In the following, each metric and the grading instructions assigned to the annotators for each metric will be described in detail.

5.5.1. Accuracy of Features (AoF). This metric assesses whether the functional features integrated into the GUI prototypes generated by the LLM correspond with the provided description. This evaluation is critical, as it determines whether the model creates a relevant and appropriate set of features for the intended GUI rather than generating a random assortment. It evaluates the understanding of the application domain by the model and the described primary purpose of the mobile page and assesses its capability to produce appropriate GUI functionalities that align with the given domain and intent accurately. Evaluators were instructed to award the maximum grade if a GUI prototype perfectly matches the description, meaning all features were relevant and correctly represented for the described page. The focus was entirely on evaluating the accuracy of the existing features within the GUI prototype. Grades were to be reduced for features not aligned with the description.

5.5.2. Quantity of Features (QoF). In the context of requirement elicitation, prototypes play a pivotal role in enhancing the communication of functional requirements. Therefore, presenting a diverse range of features in these prototypes is advantageous for fruitful discussions. This metric was introduced to determine which prompts best encourage the model to produce a wide variety of features for the described GUI. A prototype that incorporates many distinct features should be graded highly. GUI prototypes that include fewer functionalities should receive lower QoF grades.

5.5.3. Visual Appeal (VA). This metric extends beyond just functional requirement elicitation and encompasses the evaluation of the prototypes as a source of inspiration for the design and layout of the page. Instructions for this metric were kept concise, as this evaluation is more subjective and reflect the individual's personal preferences of the prototype's visual appeal.

6. Results and Discussion

The subsequent sections will provide the results of the conducted experiments and answer our research questions. The results are displayed in Figure 2 and are supplemented by statistical testing. To analyze whether the various prompting approaches yield significantly different results within a metric, we conducted pairwise Wilcoxon signed-rank tests (Wilcoxon, 1992). The applied Wilcoxon signed-rank test is a non-parametric test and therefore well-suited for our ordinal Likert scale data, as it compares two sets of related data points, in this case, metric scores on GUIs that have been generated from the same description using different prompting methods. Within this test, the difference between the observation pairs is computed first and then the absolute differences are ranked with averages ranks for ties. Subsequently, the test statistic W is computed as the minimum of the sum of ranks with earlier positive difference and sum of ranks with earlier negative difference, respectively. The effect sizes for the Wilcoxon signed-rank tests were calculated as $r = Z/\sqrt{N}$, with Z being the standardized test statistic and N the number pairs within the Wilcoxon signed-rank test, following the computation in (Rosenthal, Cooper, Hedges, et al., 1994; Fritz, Morris, & Richler, 2012). This measurement includes not only the magnitude but also the direction of the effect size, facilitating the interpretation of the results between the investigated prompting methods for prototyping.

6.1. RQ1: Comparison of Structured and Unstructured Prompting Techniques

Table 1 shows the Wilcoxon signed-rank test results for all three metrics, comparing *Instruction* with *CoT-ZS* and *Instruction* with *PD-ZS* prompting. As indicated in the box plot evaluation results from Figure 2, a noteworthy observation is a marginal difference in the average scores regarding the feature accuracy metric between the *Instruction*, *CoT-ZS*, and *PD-ZS* prompt. This minimal difference is confirmed by the Wilcoxon signed rank tests, which yielded a p-value of above 0.05 for the AoF metric. No statistically significant differences but uniformly high average scores for this metric indicate that *GPT-4* effectively comprehends the app domains and generates prototypes that align with the provided descriptions, regardless of employing instruction prompt or a structured prompting approach.

Similar to the feature accuracy metric, the use of *ZS* structured prompting does not significantly enhance the visual appeal of the GUI prototypes, as indicated again by the above statistically significant Wilcoxon signed-rank p-values. This shows that a *ZS* structured prompting approach alone does not improve the visual appeal of the generated GUI prototypes.

The quantity of features metric strongly contrasts with the AoF and VA metrics, as the difference in scores between unstructured and structured approaches is significantly greater. The Wilcoxon signed-rank tests show that compared to unstructured *Instruction* prompting, *CoT-ZS* and *PD-ZS* prompting significantly enhance the scores for the QoF metric. Hence, as a result, the scores for this metric increase from (Median=3/Mean=2.8) to (Median=3/Mean=3.1) for CoT-ZS prompting and to (Median=4/Mean=3.8) for PD-ZS. This shows, that through a structured prompting approach such as CoT-ZS and PD-ZS, the comprehensiveness of the prototypes regarding the amount of functionalities present in the GUI is enhanced significantly. Figure 3A displays a comparison between a GUI generated through Instruction prompting and CoT-ZS prompting approach, showing an enhanced GUI feature complexity via structured prompting techniques. For example, this includes additional useful features for an alarm clock settings GUI such as selection of time format and alarm sound as well as save/reset options.

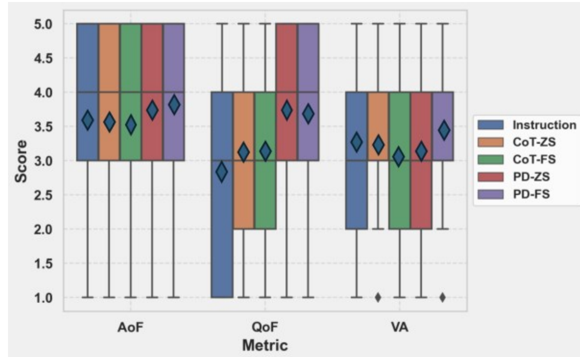

Figure 2. Box plot showing results across the five different models and the metrics Accuracy of Features (AoF), Quantity of Features (QoF) and Visual Appeal (VA) (diamonds represent means)

Table 1. p-values and effect sizes (r) for the Wilcoxon signed-rank tests between Instruction and CoT-ZS as well as Instruction and PD-ZS prompting

Metric	Inst. vs. CoT-ZS		Inst. vs. PD-ZS	
	p-value	r	p-value	r
AoF	0.98	0.5	0.15	0.45
QoF	0.001	0.41	7.37e-13	0.58
VA	0.84	0.32	0.52	0.28

6.2. RQ2: Comparison of CoT-ZS and PD-ZS

As already discussed in RQ1, the accuracy of the generated features for the prototypes is not significantly different when comparing *CoT-ZS* and *PD-ZS*. The same observation can be made about the visual appeal of the GUI prototypes, as the Wilcoxon signed-rank test does not reject its null hypothesis as shown in Table 2.

Nevertheless, the difference in median/mean scores for both approaches and the Wilcoxon signed-rank test identify further improvement regarding the number of present features from *CoT-ZS* to *PD-ZS*. Examining the complete outputs from both prompting techniques, not just the final GUI prototypes, reveals that CoT-ZS prompts tend to produce shorter feature lists than the PD-ZS approach. Consequently, this leads to prototypes encompassing a less comprehensive set of functionalities. Moreover, PD-ZS contains more standard features (not just any extra random features) that CoT-ZS might miss (e.g., language detection, voice input for a translation app as shown in Figure 3B and favorite button for list items, filter options for lists for the job search app as shown in Figure 3C). In the displayed example, the CoT-ZS prompt generated a list of six distinct features. In

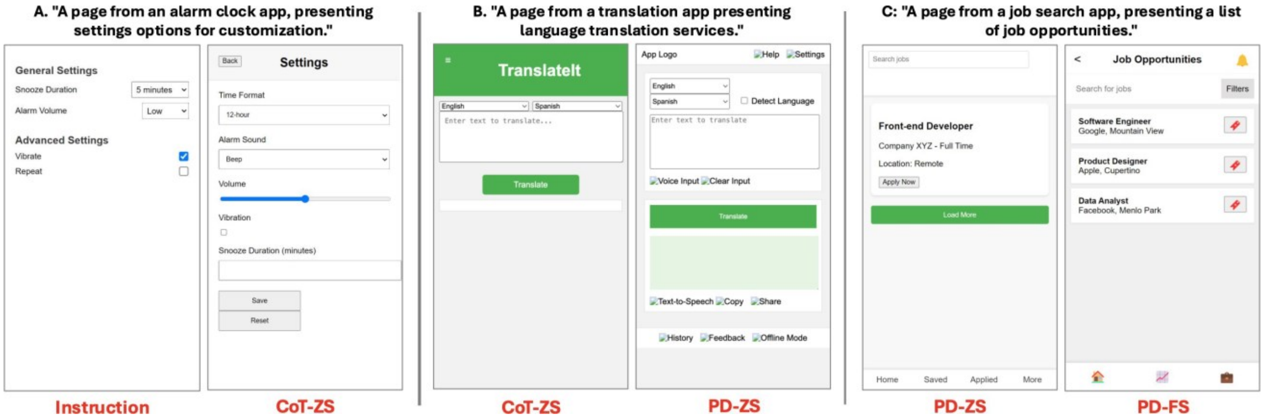

Figure 3. Comparison of generated GUI prototypes with different prompting approaches (Instruction, CoT, PD)

Table 2. p-values and effect sizes (r) for the Wilcoxon signed-rank tests between CoT-ZS and PD-ZS prompting approaches

Metric	CoT-ZS vs. PD-ZS	
	p-value	r
AoF	0.08	0.49
QoF	2.56e-10	0.54
VA	0.79	0.32

Table 3. p-values and effect sizes (r) for the Wilcoxon signed-rank tests between ZS and FS.

Metric	CoT-ZS vs. CoT-FS		PD-ZS vs. PD-FS	
	p-value	r	p-value	r
AoF	0.34	0.57	0.16	0.5
QoF	0.87	0.35	0.21	0.4
VA	0.11	0.36	0.01	0.42

comparison, the *PD-ZS* approach generated a significantly more comprehensive list of eleven features, resulting in a higher comprehensive score of 5 for the quantity of features, compared to the score of 3 from the *CoT-ZS*-generated GUI prototype.

6.3. RQ3: Comparison of Zero-Shot and Few-Shot Prompting Techniques

Also, when it comes to the comparison of zero-shot and few-shot prompting, no significant difference in the accuracy of features can be observed through the observation of the mean scores and by the Wilcoxon signed-rank-test results in Table 3.
In addition, the prototypes generated from the few-shot prompts show no improvement in their feature quantity score compared to their zero-shot counterparts. This is underlined by the Wilcoxon signed-rank test which null hypothesis cannot be rejected. Furthermore, when observing the comparison of *CoT-ZS* and *CoT-FS*, enhancing the prompt with input-output examples in the *CoT* approach does not significantly impact the visual appeal of the GUI. However, the GUI prototypes receive significantly higher ratings when the *PD* approach incorporates few-shot prompting, boosting the respective scores from (*Median*=4/*Mean*=3.2) to (*Median*=4/*Mean*=3.5). An improvement from zero-shot to few-shot prompting is expected, given that the model is supplied with two real-world examples featuring fully comprehensive GUI prototype designs.

We observed that the *PD-FS* prompt generates GUI prototypes with several characteristics that could be a reason for enhanced visual appeal. These prototypes exhibit a more diverse use of colors, a significantly larger incorporation of icons, and fewer flaws related to the layout and spacing, resulting in a much cleaner overall design. Figure 3C displays a comparison of two GUIs generated through the *PD-ZS* and *PD-FS* prompt.

7. Limitations

The evaluation highlights *GPT-4's* ability to create high-quality GUI prototypes when paired with proper prompts. However, a notable limitation of this approach is the model's occasional inconsistency, leading to unexpected or unwanted outcomes. Although such inconsistencies appear infrequently, their presence, also when utilizing the best-performing prompting techniques, underscores that the results generated by the LLM are not entirely reliable. Three main types of inconsistencies were identified. First, *missing mobile layout*: In several cases, the model overlooked explicit instructions to create mobile app layouts, resulting in GUIs suited for web page layouts but inappropriate for mobile applications. Second, *incomplete feature implementation*: Despite often yielding highly detailed prototypes, *GPT-4* produced GUIs with features that were not fully implemented in some instances. These included plain text descriptions as placeholders instead of complete GUI components. Third, *alignment issues*:

While this might be considered a minor issue and is easily correctable manually, the model occasionally mishandles the alignment of UI elements. This can lead to features being visually obscured by others, making them invisible within the prototypes. In some cases, UI components extended beyond the mobile app layout or were even entirely placed outside the intended layout.

8. Threats to Validity

Internal Validity. The primary challenge to internal validity in our experiments arises from the involvement of human evaluators to assess the quality of the generated GUI prototypes. This introduces potential user biases and the risk of misunderstanding or misinterpreting the evaluation metrics. To mitigate user bias, we provided the prototypes without indicating the prompts that generated them (i.e. randomized). Additionally, we provided detailed descriptions of our metrics and the annotation task to minimize misinterpretations. It is worth mentioning that we used an LLM to synthesize functional GUI requirements. Typically, humans would write these instructions, which could significantly differ from the consistently well-structured ones created by the LLM. As a result, the LLM-generations might be easier to process than human-generated content, which can be less precise and more ambiguous. However, recent research has shown that LLMs are capable of producing accurate GUI descriptions (Wang et al., 2023) and understand detailed functional semantics of GUIs (Yan et al., 2023; Kolthoff et al., 2024). To enable the reproducibility of our experiments, we have released the code and all prompts that were used in the evaluation.

External Validity. The evaluation dataset is relatively small, including only 30 entries. This limited scope may not sufficiently represent the vast diversity of real-world GUIs. However, in our sampling of *Rico* GUIs, we aimed to encompass a wide range of application domains and mobile screen functionalities. As a result, each described GUI in our dataset is distinct regarding its main functionality and spans across 21 unique app domains. Furthermore, the results obtained with the employed LLM may not generalize to other LLMs. Any assertions regarding the feasibility of enhancing GUI prototyping of LLMs with the presented techniques apply specifically to the employed *GPT-4* model, but we plan to conduct further evaluation with additional popular LLMs in the future. However, we particularly focused on employing *GPT-4* due to its SOTA performance (Achiam et al., 2023) to explore enhancements in GUI prototyping, noting that for less effective LLM alternatives it might be easier to show GUI prototyping improvements through presented prompting techniques. To increase reproducibility of our results, we provided all LLM settings and employed a reduced temperature as a trade-off with creativity.

9. Conclusion

In this paper, we presented prompting techniques to enhance the generation of GUI prototypes from high-level text descriptions using an LLM. We evaluated the influence of a structured prompting strategy to guide the LLM through the prototyping process, examined the enhancement of *CoT* reasoning through *PD*, and the potential of few-shot prompting. Using *PD*, *GPT-4* produced detailed, accurate, clear, and useful GUI prototypes while few-shot prompting could additionally improve the visual appeal of generated GUI prototypes. The proposed approach could be integrated by analysts during initial requirements elicitation interviews with customers to rapidly create initial GUI prototypes to directly obtain customer feedback. Furthermore, the generated GUI prototype representation could be integrated into prototyping tools rendering them editable, to adapt them according to customer feedback.

References

Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman F. L., . . . others (2023). Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Adobe xd. (2024). https://helpx.adobe.com/support/xd.html. (Accessed: 02/21/2024)

Aldave, A., Vara, J. M., Granada, D., & Marcos, E. (2019). Leveraging creativity in requirements elicitation within agile software development: A systematic literature review. *Journal of Systems and Software*, *157*, 110396.

Behrang, F., Reiss, S. P., & Orso, A. (2018). Guifetch: supporting app design and development through gui search. In *Proceedings of the 5th international conference on mobile software engineering and systems* (pp. 236– 246).

Beltramelli, T. (2018). pix2code: Generating code from a graphical user interface screenshot. In *Proceedings of the acm sigchi symposium on engineering interactive computing systems* (pp. 1–6).

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., . . . others (2020). Language models are few-shot learners. *Advances in neural information processing systems*, *33*, 1877–1901.

Bunian, S., Li, K., Jemmali, C., Harteveld, C., Fu, Y., & Seif El-Nasr, M. S. (2021). Vins: Visual search for mobile user interface design. In *Proceedings of the 2021 chi conference on human factors in computing systems* (pp. 1–14).

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J., . . . others (2021). Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Coyette, A., Faulkner, S., Kolp, M., Limbourg, Q., & Vanderdonckt, J. (2004). Sketchixml: towards a multi-agent design tool for sketching user Interfaces based on usixml. In *Proceedings of the 3rd annual conference on task models and diagrams* (pp. 75–82).

Deka, B., Huang, Z., Franzen, C., Hibschman, J., Afergan, D., Li, Y., . . . Kumar, R. (2017). Rico: A mobile app dataset for building data-driven design applications. In *Proceedings of the 30th annual acm symposium on user interface software and technology* (pp. 845–854).

Dong, Q., Li, L., Dai, D., Zheng, C., Wu, Z., Chang, B., . . . Sui, Z. (2022). A survey for in-context learning. *arXiv preprint arXiv:2301.00234*.

Douglas, B. D., Ewell, P. J., & Brauer, M. (2023). Data quality in online human-subjects research: Comparisons between mturk, prolific, cloudresearch, qualtrics, and sona. *Plos one*, *18*(3), e0279720.

Eloundou, T., Manning, S., Mishkin, P., & Rock, D. (2023). Gpts are gpts: An early look at the labor market impact potential of large language models. *arXiv preprint arXiv:2303.10130*.

Figma. (2024). https://www.figma.com. (Accessed: 02/21/2024)

Fritz, C. O., Morris, P. E., & Richler, J. J. (2012). Effect size estimates: current use, calculations, and interpretation. *Journal of experimental psychology: General*, *141*(1), 2.

gpt-4-1106-preview. (2024). https:// platform. openai.com/docs/models/continuous-model-upgrades. (Accessed: 01/27/2024)

Huang, F., Li, G., Zhou, X., Canny, J. F., & Li, Y. (2021). Creating user interface mock-ups from high-level text descriptions with deep-learning models. *arXiv preprint arXiv:2110.07775*.

Huang, Z. (2020). *Deep-learning-based machine understanding of sketches: Recognizing and generating sketches with deep neural networks* (Unpublished doctoral dissertation). Master's thesis, University of California at Berkeley.

Kojima, T., Gu, S. S., Reid, M., Matsuo, Y., & Iwasawa, Y. (2022). Large language models are zero-shot reasoners. *Advances in neural information processing systems*, *35*, 22199–22213.

Kolthoff, K., Bartelt, C., & Ponzetto, S. P. (2020). Gui2wire: rapid wireframing with a mined and large-scale gui repository using natural language requirements. In *Proceedings of the 35th ieee/acm international conference on automated software engineering* (pp. 1297–1301).

Kolthoff, K., Bartelt, C., & Ponzetto, S. P. (2023). Data-driven prototyping via natural-language-based gui retrieval. *Automated Software Engineering*, *30*(1).

Kolthoff, K., Kretzer, F., Bartelt, C., Maedche, A., & Ponzetto, S. (2024, jun). Interlinking user stories and gui prototyping: A semi-automatic llm-based approach. In *2024 ieee 32nd international requirements engineering conference (re)* (p. 380-388). Los Alamitos, CA, USA: IEEE Computer Society. Retrieved from https://doi.ieeecomputersociety.org/10.1109/RE59067.2024.00045 doi: 10.1109/RE59067.2024.00045

Li, T. J.-J., Popowski, L., Mitchell, T., & Myers, B. A. (2021). Screen2vec: Semantic embedding of gui screens and gui components. In *Proceedings of the 2021 chi conference on human factors in computing systems* (pp. 1–15).

Liu, P., Yuan, W., Fu, J., Jiang, Z., Hayashi, H., & Neubig, G. (2023). Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys*, *55*(9), 1–35.

Palan, S., & Schitter, C. (2018). Prolific. ac—a subject pool for online experiments. *Journal of behavioral and experimental finance*, *17*, 22–27.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. (2019). Language models are unsupervised multitask learners. *OpenAI blog*, *1*(8), 9.

Repository. (2024). https://github.com/ gui- prototyping-with-llms/ effective-gui- prototyping. (Accessed: 09/03/2024)

Requirements engineering processes, tools/technologies, & methodologies. (2009).

Rocha, T., Hak, J.-L., Winckler, M., & Nicolas, O. (2019). A comparative study of milestones for featuring gui prototyping tools. *arXiv preprint arXiv:1906.01417*.

Rosenthal, R., Cooper, H., Hedges, L., et al. (1994). Parametric measures of effect size. *The handbook of research synthesis*, *621*(2), 231–244.

Shanahan, M. (2024). Talking about large language models. *Communications of the ACM*, *67*(2), 68–79.

Sonje, S., Dave, H., Pardeshi, J., & Chaudhari, S. (2022). draw2code: Ai based auto web page generation from hand-drawn page mock-up. In *2022 ieee 7th international conference for convergence in technology (i2ct)* (pp. 1–7).

Tiwari, S., Rathore, S. S., & Gupta, A. (2012). Selecting requirement elicitation techniques for software projects. In *2012 csi sixth international conference on software engineering (conseg)* (pp. 1–10).

Wang, B., Deng, X., & Sun, H. (2022). Iteratively prompt pre-trained language models for chain of thought. In *Proceedings of the 2022 conference on empirical methods in natural language processing* (pp. 2714–2730).

Wang, B., Li, G., & Li, Y. (2023). Enabling conversational interaction with mobile ui using large language models. In *Proceedings of the 2023 chi conference on human factors in computing systems* (pp. 1–17).

Wang, B., Li, G., Zhou, X., Chen, Z., Grossman, T., & Li, Y. (2021). Screen2words: Automatic mobile ui summarization with multimodal learning. In *The 34th annual acm symposium on user interface software and technology* (pp. 498–510).

Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., . . . others (2022). Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E. H., . . . others (2022). Chain-of-thought prompting elicits reasoning in large language models. In *Advances in neural information processing systems.*

Wilcoxon, F. (1992). Individual comparisons by ranking methods. In *Breakthroughs in statistics: Methodology and distribution* (pp. 196–202). Springer.

Yan, A., Yang, Z., Zhu, W., Lin, K., Li, L., Wang, J., . . . others (2023). Gpt-4v in wonderland: Large multimodal models for zero-shot smartphone gui navigation. *arXiv preprint arXiv:2311.07562*.

Zhou, D., Scha¨rli, N., Hou, L., Wei, J., Scales, N., Wang, X., . . . others (2022). Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*.