DynEx: Dynamic Code Synthesis with Structured Design Exploration for Accelerated Exploratory Programming

Jenny Ma
jenny.ma@columbia.edu
Columbia University
New York, New York, USA

Karthik Sreedhar
ks4190@columbia.edu
Columbia University
New York, New York, USA

Vivian Liu
vivian@cs.columbia.edu
Columbia University
New York, New York, USA

Sitong Wang
sw3504@columbia.edu
Columbia University
New York, New York, USA

Pedro Alejandro Perez
pap2153@columbia.edu
Columbia University
New York, New York, USA

Riya Sahni
rs4640@columbia.edu
Columbia University
New York, New York, USA

Lydia B. Chilton
chilton@cs.columbia.edu
Columbia University
New York, New York, USA

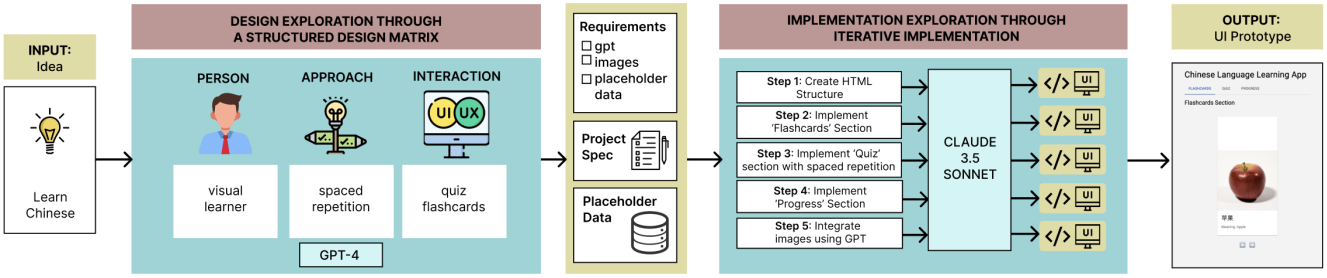

Figure 1: DynEx is an LLM-based system for exploratory programming. It guides users through a design space using a structured Design Matrix. Based on the Design Matrix content, DynEx brainstorms the necessary requirements needed to build the UI, generates a detailed project spec to translate the abstract concepts into an implementation plan, and generates synthetic placeholder data to better prototype the application. DynEx then breaks down the project into tasks, which users can iteratively implement until they create a working prototype. Users can incrementally add features until they are satisfied, and also create a variety of new prototypes using the system.

Abstract

Recent advancements in large language models have significantly expedited the process of generating front-end code. This allows users to rapidly prototype user interfaces and ideate through code, a process known as exploratory programming. However, existing LLM code-generation tools focus more on technical implementation details rather than finding the right design given a particular problem. We present DynEx, an LLM-based method for design exploration in accelerated exploratory programming. DynEx uses LLMs to guide users through a structured Design Matrix to explore the design space before dynamic iterative implementation. It also introduces a technique to self-invoke generative AI, enabling the creation of a diverse suite of applications. A user study of 10 experts found that DynEx increased design exploration and enabled the creation of more complex and varied prototypes compared to a Claude Artifact baseline. We conclude with a discussion of the implications of design exploration for exploratory programming.

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

CCS Concepts

• **Human-centered computing** → **User interface programming**.

Keywords

code synthesis, exploratory programming, design exploration, design matrix, user interface, prototyping

ACM Reference Format:
Jenny Ma, Karthik Sreedhar, Vivian Liu, Sitong Wang, Pedro Alejandro Perez, Riya Sahni, and Lydia B. Chilton. 2018. DynEx: Dynamic Code Synthesis with Structured Design Exploration for Accelerated Exploratory Programming. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 18 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Recently released large language models (LLMs) have shown remarkable capabilities in generating code, particularly front-end code [35]. This allows users to get ideas off the ground faster by building code-based user-interface (UI) prototypes and testing them directly in real world contexts, a process known as exploratory programming. Exploratory programming is crucial for experimental projects where real design testing is more productive than upfront specifications. Many interactions are difficult to simulate in low-fidelity prototyping mediums, particularly when testing data-driven applications [4]; prototyping is crucial in these cases to truly test ideas. LLM code-generation abilities present a unique opportunity to accelerate exploratory programming and ideate through code.

When prototyping, however, translating abstract concepts into concrete implementations is a challenging process [40]. There is a gap in going from an idea to a working solution because initial ideas often lack clear boundaries, structured organization, and concrete details. Converting an initial idea into a feasible design requires a detailed consideration of multiple aspects: users must fully consider the problem being solved, the target user of the application, the approach or methodology, and the interaction paradigm or user experience [3]. Each of these elements must be defined before moving forward with implementation, making the transition from idea to design a complex and multifaceted process.

Existing tools that leverage LLM code-generation capabilities, such as GPTPilot [32] and OpenHands (formerly OpenDevin) [43], predominantly focus on code development rather than creating an application that considers the end-user experience. GPTPilot prompts the user to consider implementation-level details, such as which packages to use, rather than broader aspects of the user's ideas, like who the application is for, what problem the application is trying to solve, and what the core methodology is for guiding the solution. Claude Artifact is more suitable for exploration as it enables users to chat with the LLM, but its fundamentally unstructured approach can also be limiting. Users easily become fixated on implementation over exploration, appending features onto prototypes linearly rather than using the chat-bot to laterally exploring their problem space [2]. Ultimately, emphasis on implementation over user-centric design can lead to design fixation [1]; users may be led down one path of technical implementation, but lack the divergent exploration that can enable them to find the most fitting approach for their problem [8]. These applications may function well technically, but can lack the depth of refinement that helps ground a prototype in real user needs.

In exploratory programming, it is crucial to explore different solutions before implementation [34]. Our approach is to integrate design with code synthesis. Before implementing a solution, we consider many factors, such as the user (who are they and why do they want this?), the core approach (for a learning application, are you following a learning theory like spaced repetition or generation and elaboration?), and the interaction paradigm (if you are building a crowd-sourced hotel search platform, are you basing it off of an existing paradigm like Uber, or a card-swipe paradigm like Tinder for low cognitive load?). By providing a structured framework for design exploration, we can guide users to collaborate with LLMs in exploring a problem space. Combined with their code-generation capabilities, LLMs can assist users in creating user-centric, code-based UIs that serve as both functional prototypes and stepping stones towards final products.

We present DynEx, an LLM-based method for exploratory programming for functional UIs. DynEx helps the user build code-based UI prototypes in two stages: (1) structured design exploration through a Design Matrix, and (2) dynamic iterative implementation with LLM code synthesis. The Design Matrix brainstorms unique ideas through through idea generation to help users explore the design space. The idea is then grounded, a process to synthesize abstract ideas into detailed application designs. The system next breaks down the project into steps to execute iteratively, and generates code that self-invokes multi-modal LLMs to produce applications that have larger sets of synthetic placeholder data, can generate images, and provide recommendations, enabling the creation of a diverse suite of applications.

Our contributions are as follows:

- DynEx, an LLM system for accelerated exploratory programming where the user inputs a problem they want to solve and rapidly creates UI prototypes. The system guides users through a design space, then allows users to iteratively generate and interact with UI code. Users can create multiple variations of prototypes and iteratively refine their ideas.
- the Design Matrix, a structured framework using LLMs to guide users through the design space by considering the person, approach, and interaction through idea generation and grounding.
- Self-invoked multi-modal LLMs, a technique for building code-based UI prototypes where the generated code can call other LLMs to create a diverse and rich suite of applications.
- A qualitative evaluation of 10 experts, demonstrating that DynEx enabled them to explore the design space and create more complex and varied prototypes as compared to a Claude Artifact baseline where users must lead the design process themselves.

2 Related Works

2.1 Exploratory Programming and Prototyping

Exploratory programming is a practice in which programmers actively experiment with different possibilities using code [4]. It is grounded by 5 main characteristics: (1) Needs for Exploration, (2) Code Quality Tradeoffs, (3) Ease or Difficulty of Exploration, (4) Exploration Process, and (5) Group or Individual Exploration. Like other prototyping methods, it is important to consider the cost of exploratory programming compared to its value in providing insights [39]. Exploratory programming for UI creation must be as easy and quick as possible while still creating functional code.

It must also produce sufficiently complex interfaces that can actually inform designers on how users will interact with them. Finally, it is crucial that exploratory programming processes support lightweight version control so that individuals can easily see the variations they create [19].

Prototyping and exploratory programming are forms of creative problem-solving. During exploratory programming, designers iteratively test and refine ideas via code. The ability to generate multiple alternatives, explore directions, and revert to previous versions have been labeled as requirements for creativity support tools [34]. Past research demonstrates the effectiveness of prototyping during design [8, 9]. Learning theory research suggests that generating variations of outputs enables critical reflection and deepens understanding in problem domains [25]. Prototyping has been demonstrated to be effective in serving as a means of inquiry to increase granularity of design requirements based on user reactions [18, 46]. However, it is not always possible to prototype UIs in low-fidelity mediums. Applications often change requirements and need to support complex user interactions [6]. Even higher-fidelity approaches like Figma [10] also have limitations in modeling the interactive experience of a UI. It can lack the richness of user experience that comes from even simple backend computation or data processing [4] and have other time and resource costs [39]. Exploratory programming addresses these issues, allowing for the prototyping of code-based interactive applications.

2.2 LLM-Based Tools for Code Generation

2.2.1 LLM-Based Code Synthesis Tools. State of the art models for code generation include GPT-4, AlphaCode, CodeGEN, Code Llama, and Gemini [13, 21, 26, 29, 30]. These models generally perform well in transforming a natural language problem specification into a simple code solution. They are generally most effective at creating code segments for specific functions or features. Systems built around these LLMs show promise for improving LLM-code generation beyond direct prompting. However, many of the UI code generation tools have limitations. For example, Co-Pilot [11] is a system that can modify existing code repositories, but it does not support end-to-end code generation. OpenHands [43] and GPTPilot [32] are open-source agent driven systems for end-to-end software development. They use separate LLM agents to represent various roles within the software development process and implement a technical specification. However, these systems lack the support for exploratory programming that is embedded in DynEx, which helps users explore during the planning and problem specification stages of development.

2.2.2 LLM-Based Code Synthesis Tools for UIs. LLMs have been proven to be especially good at creating UI code. Specifically, LLMs have been shown to be extremely apt at writing code from provided UI screenshots and designs [23, 33, 35, 36, 42, 47, 50, 52]. LLMs have also shown the ability to write UI code from natural language prompts; WebSim is a system which enables users to created simulated UIs with minimal prompting [44]. However, these systems do not offer users much design support, placing the brunt of ideation and design space exploration on the user. DynaVis is a tool that dynamically synthesizes widgets for data visualization UI. While it allows users to make dynamic components, it does not support the creation of a complete user interface [41].

Claude Artifact [2] likely offers the best out-of-the-box solution for UI prototyping. It creates functional UIs from conversational prompts, provides users with a window for rendering, and keeps track of version history. However, Claude Artifact does not guide users through the design exploration process. The onus is on the user to use repeated prompting to add new features and move towards a solution. Given that LLMs have demonstrated the ability to effectively power creativity support tools and perform divergent design space exploration in other domains [31, 37, 51], it should be possible to build LLM systems that support both design exploration and implementation for UI prototyping.

2.3 Informing the Development of LLM-Based Creativity Support Tools

It is necessary for LLM-Based creativity support tools to have sufficient structures for users to create and compare variations within the design space [34]. Moreover, users need to be able to think through all dimensions of their problem, which is a challenging task for people to do on their own [24]. When left to their own devices, people tend to prematurely converge on ideas before thinking through all possibilities [7, 9, 15, 16].

Prior work enables thorough design space exploration by helping users define dimensions to explore. Luminate [37] is a system for exploring design spaces for narratives based on dimensions that can characterize the design space. Users can arrange and cluster generated outputs based on dimensions such as genre, tone, or setting to structure the design space. Other LLM-based writing support tools use pre-defined dimensions such as feasibility and usability, as opposed to synthesizing dimensions on the fly [12]. We realize this dimensional thinking in DynEx's Design Matrix. By having users traverse through the Design Matrix before any implementation occurs, DynEx forces users to thoroughly think through all of the dimensions of the problem.

Self-referential multi-modal LLMs have been demonstrated to improve the outputs of LLM-based systems. WebSim [44] utilizes self-referential LLMs to generate images for created UIs. Jigsaw is a system for prototyping that utilizes puzzle pieces as metaphors to represent AI foundational models [22]. This representation allows users to understand and chain foundational model capabilities for the creation of more complex outputs. We build upon self-referential LLMs in DynEx by self-invoking methods that can allow for the creation of images or placeholder data, and emulate recommendation or sensemaking systems to power the UI prototypes.

There are several other factors we consider more broadly that are relevant to LLM-systems. First, many tasks are too complex for LLMs to accomplish in one prompt and have to be broken down into smaller problems [48, 49]. LLMs are also more successful in accomplishing tasks when asked to create a plan and reason through intermediate steps [5, 20, 38, 45]. Creating a complete UI is clearly a involved task; DynEx breaks down the project into steps to create an implementation plan that can be iteratively executed. Second, tasking non-AI experts with writing many trivial prompts can reduce the effectiveness of LLM-systems [5, 53]. In prior prototyping studies, users spent excessive time debugging and

iterating on prompts [17]; thus, DynEx has structured prompting that enables users to focus their energy on expressing their ideas as opposed to prompt engineering for functional outputs. Finally, the difficulty in sourcing example data has limited the success in prototyping with LLMs. [17]. DynEx therefore has specific functionality to self-invoke LLMs and easily create synthetic data for applications.

3 System

We present DynEx, a LLM-based system for accelerated exploratory programming that assists users with design exploration and code implementation. The input is the problem that the user wants to solve. It can be vague idea, such as an application to help with plant-watering or meal-planning; something that the user does not have a clear solution for. The output is a functional UI that can support placeholder data, images, and dynamic data. It is an HTML page written in Javascript, HTML, and CSS, that also utilizes React [27] and Material UI Library [28], a comprehensive React component library offering foundational elements like buttons, dropdowns, input forms, and more.

DynEx is designed for programmers who have personal projects and unrealized ideas they want to explore. Many of these ideas remain underdeveloped or unrealized for years, if not indefinitely. Bringing these ideas to fruition is challenging because it not only requires developer experience to prototype the idea in code, but also requires addressing gaps in the design. DynEx helps concretize users' ideas through design exploration and implementation, allowing them to take a crucial step towards creating a minimal viable product (MVP).

The pipeline has 2 steps:

(1) **Design Exploration through the Design Matrix**, where the system helps the user identify the person, approach, and interaction paradigm of the application using idea generation and grounding, a method of specifying ideas to become a more solidified design.
(2) **Dynamic Iterative Implementation**, where the system generates code iteratively, allowing for user input at every step, until the creation of the final prototype. Our implementation process supports the generation of synthetic placeholder data and self-invokes multi-modal LLMs, a technique for the generated code to call other LLMs to create a dynamic and diverse suite of applications.

The system was written in Python, Typescript, and Flask. We use GPT-4 in our Design Matrix for design exploration – due to its vast understanding across various domains, it is well suited for brainstorming and refining ideas. We use Claude 3.5 Sonnet for code generation – Claude is known to be better at writing UI code, and it enables us to use Claude Artifact as a baseline for comparison in our evaluation. Claude is also highly responsive to explicit instructions, which is crucial for code-generation.

3.1 Design Exploration through the Design Matrix

A user begins by suggesting a problem they would like to solve, such as "I need reminders water my plants" or "I need to plan my meals"– it can be vague, something they have not thought of a solution for. Once submitted, the user is led through the Design Matrix, a structured framework that guides users through the design space. The matrix is centered on three key dimensions that are foundational to user-centric design, which make up the columns:

(1) Person - who the application is for.
(2) Approach - the method, theory, or strategy behind the solution.
(3) Interaction - the core user interaction paradigm, such as a table, card-swipe, news feed, or chat-bot layout.

Each dimension is explored on two levels of specificity, which make up the rows of the matrix:

(1) Idea: the brainstorming of the higher-level concepts across each dimension
(2) Grounding: the solidification of foundational details through iterative deepening that specifies how the idea should be developed. For example, if the user selected a card-swipe UI as the interaction paradigm, the grounding should specify what information exists on each card, what swiping right means, and what swiping left means.

Fig. 2 illustrates the Design Matrix, with columns for Person, Approach, and Interaction, and rows for Idea and Grounding. The Design Matrix facilitates the transformation of abstract concepts into actionable design components, allowing the user to fully understand their problem and solution space before settling on a prototype design.

3.1.1 Design Matrix Interaction. Users can traverse through the Design Matrix in any order, as long as they submit the Idea before the Grounding level for each dimension. On the Idea level of specificity, the system help users brainstorm different ideas through divergent thinking; the Grounding level of specificity solidifies the idea through convergent thinking. To reference entries in the matrix, we use a *column:row* notation; for example – *Person:Idea*, *Approach:Grounding*.

In order to ground the idea, the idea must be already submitted for that particular dimension; *Person:Idea* must be submitted before *Person:Grounding*, *Approach:Idea* must be submitted before *Approach:Grounding*, and *Interaction:Idea* must be submitted before *Interaction:Grounding*. Otherwise, the user can brainstorm the Person, Approach, and Interaction dimensions in any order. For example, if the user has the application idea of "tinder for groceries", the interaction paradigm is clear – they want a card-swipe UI. They can fill out the Interaction dimension first before formalizing the Person and the Approach. If the user wants to build a journalling application to help with mental health that uses cognitive behavioral therapy (CBT) principles, they can fill out the Approach dimension to use CBT before filling out the Person and Interaction dimensions.

We use context curation to guide the outputs of each entry in the matrix. The system takes in all previously-submitted entries of the matrix, since they are already-known aspects of the project, to help inform the response of the current entry. The previous entries used as context are highlighted in yellow on the UI for the user to visualize (see Fig. 2 - I). If the user chooses to resubmit a dimension on the Idea level, the Grounding for that dimension will not be factored in the context. Responses for each entry were generated using few-shot examples and can be found in Appendix 8.

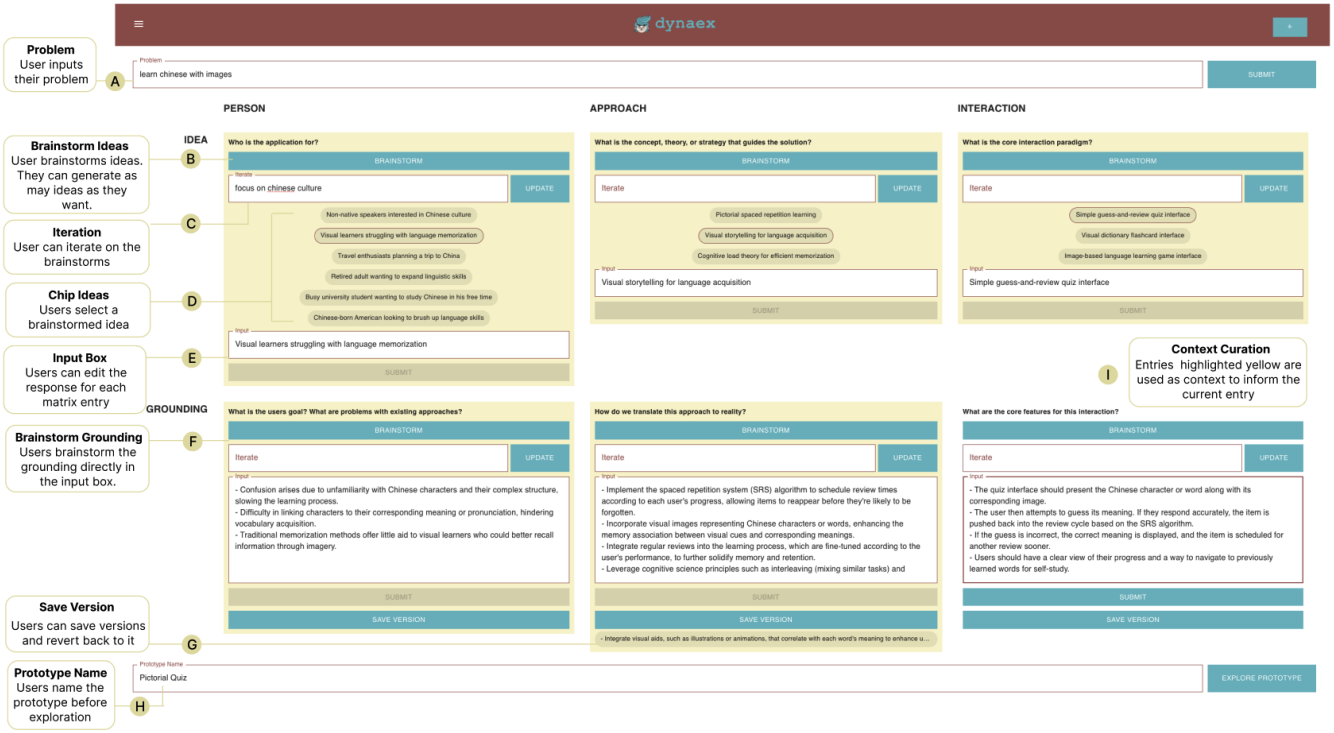

Figure 2: DynEx's Design Matrix User Interface: Users traverse through the matrix, which guides them through exploring the Person, Approach, and Interaction dimensions relevant to the problem space. For each dimension, users begin on the Idea level by inputting their problem (A). They then brainstorm ideas (B), and can iterate on the ideas (C). They can select an idea (C) and submit it in the input box (E). They then move on to the Grounding level, brainstorm a response (F), and submit that as well. They have the option to save versions (G), in case they want to move back to the Idea level and explore a new idea. Previous entries used to curate the current entry response are highlighted in yellow. Finally, users can can explore the prototype (H).

The next sections dive into the specifics of each matrix entry, using a motivating example to illustrate the process. The user inputs a problem they want to solve: learn Chinese.

3.1.2 Person. The user collaborates with the system to brainstorm who the application is for in *Person:Idea*. The system brainstorms potential target users and returns 3 results:

(1) *"Non-native speakers interested in Chinese culture"*
(2) *"Visual learners struggling with language memorization"*
(3) *"Travel enthusiasts planning a trip to China"*

Each of these cases have nuances; for (1) the user would probably be more interested in creating an app to learn Chinese culture rather than Chinese words; for (2), the user prefers a visual learning application; for (3), the user would probably focus more on learning travel-related vocabulary and phrases.

The user can generate as many ideas as they want, iterate on them and re-brainstorm, or directly update the input box. For example, if the user wanted more results that focused on learning Chinese culture, they could type that in the iteration box and brainstorm ideas along that direction. The user chooses to brainstorm more ideas, and 3 more ideas are added to the list:

(4) *"Retired adult wanting to expand linguistic skills"*
(5) *"Busy university student wanting to study Chinese in his free time"*
(6) *"Chinese-born American looking to brush up language skills"*

All of these results have different implications as well. For (4), the retired adult presumably has more time to learn Chinese – their method for learning would be different than (5), the busy university student who can only study at short intervals. Additionally, (6) presents a different use-case, a user who already has existing Chinese knowledge. Ultimately, the user selects (2) *"Visual learners struggling to learn with language memorization"*. The user moves on to *Person:Grounding* to iteratively deepen their idea.

When grounding the Person dimension, the system aims to fully understand the user goals, challenges, and the broader context in which the problem should operate. *Person:Grounding* also defines the specific problems that users face, some shortcomings of existing solutions, and how the application would address these gaps. It takes in *Person:Idea* as context, which turns yellow on the interface. The system presents a list of bullet points that deepen the idea:

- *"Confusion arises due to unfamiliarity with Chinese characters and their complex structure, slowing the learning process"*
- *"Difficulty in linking characters to their corresponding meaning or pronunciation, hindering vocabulary acquisition"*

- *"Traditional memorization methods offer little aid to visual learners who could better recall information through imagery"*

The user can directly edit, save this version, or iterate on it. Once they are satisfied, they can submit their result and move on to the Approach dimension.

3.1.3 Approach. In the Design Matrix entry for *Approach:Idea*, the system brainstorms methods or strategies that the application could employ to solve the problem, given the existing context of the matrix. Since *Person:Idea* and *Person:Grounding* were submitted, those entries are highlighted yellow and used to inform the *Approach:Idea* response. The system presents 3 Approach ideas:

(1) *"Pictorial spaced repetition learning"*
(2) *"Visual storytelling for language acquisition"*
(3) *"Cognitive load theory for efficient memorization"*

The user can generate as many ideas as they want, iterate on them and re-brainstorm, or directly update the input box. The user is intrigued by (1) *"Pictorial spaced repetition learning"*, and (2) *"Visual storytelling for language acquisition"*. They select (2) *"Visual storytelling for language acquisition"* and move on to *Approach:Grounding* to explore more details.

In *Approach:Grounding*, the system concentrates on the practical implementations by defining the essential components and logic required of the chosen approach. It takes in the existing matrix as context - *Person:Idea*, *Person:Grounding*, and *Approach:Idea*. For (2) *"Visual storytelling for language acquisition"*, the system returns:

- *"Integrate visual aids, such as illustrations or animations, that correlate with each word's meaning to enhance understanding and recall"*
- *"Develop an algorithm that links related words and images together in a meaningful story, promoting stronger memory associations"*
- *"Make use of GPT to generate context-rich sentences or mini-stories, helping to create a narrative around each word or character"*
- *"Ensure design of the learning material caters to visual learners, with a focus on vibrant, engaging, and contextually relevant graphical representations"*

The user feels inspired by this, and saves it using the "save version" button in case they want to come back to it (see Fig. 2 - G). The user goes back to *Approach:Idea* and selects (1) *"Pictorial spaced repetition"*, then moves back to *Approach:Grounding*. The system generates this:

- *"Implement the spaced repetition system (SRS) algorithm to schedule review times according to each user's progress, allowing items to reappear before they're likely to be forgotten"*
- *"Incorporate visual images representing Chinese characters or words, enhancing the memory association between visual cues and corresponding meanings"*
- *"Integrate regular reviews into the learning process, which are fine-tuned according to the user's performance, to further solidify memory and retention"*
- *"Leverage cognitive science principles such as interleaving (mixing similar tasks) and retrieval practice (recalling an item from memory), to increase learning effectiveness"*

The user decides (1) *"Pictorial spaced repetition"* is more suitable for their current needs, and moves on to the Interaction dimension.

3.1.4 Interaction. In the Design Matrix entry for *Interaction:Idea*, the system brainstorms the high-level interaction paradigm of the application, such as a swipe interface or chatbot. In this example it takes in the Person and Approach dimensions as context. The system returns

- *"Simple guess-and-review quiz interface"*
- *"Visual dictionary flashcard interface"*
- *"Image-based language learning game interface"*

The user selects (1) *"Simple guess-and-review quiz interface"* as the idea and moves on to *Interaction:Grounding*. In *Interaction:Grounding*, the system specifies the features and details of the UI components. It determines what information will be displayed, the nature of user interactions, and how these elements will contribute to the overall user experience. The grounding is provided:

- *"The quiz interface should present the Chinese character or word along with its corresponding image"*
- *"The user then attempts to guess its meaning. If they respond accurately, the item is pushed back into the review cycle based on the SRS algorithm"*
- *"If the guess is incorrect, the correct meaning is displayed, and the item is scheduled for another review sooner"*
- *"Users should have a clear view of their progress and a way to navigate to previously learned words for self-study"*

The user is happy with this and submits it. Now that the Design Matrix has been traversed, they can begin implementation in code.

3.2 Dynamic Iterative Implementation With LLM Code-Generation

DynEx leverages LLM code-generation capabilities to create the prototype. DynEx self-invokes multi-modal LLMs to enable the creation of a rich suite of applications. While prototypes should be simple, they need a minimum level of complexity and detail in order to properly emulate the user experience. Applications with images should be prototyped with images, applications that require data need relevant data, and applications that require dynamic data, such as personalization or recommendation apps, should be able to replicate that. Because our generated code calls generative AI, we enable the creation of rich applications that can sufficiently prototype the user experience.

The system first identifies the project requirements, creates the project specification (spec), and generates synthetic placeholder data if required (see Fig 3- C, D E). Once the initial setup is complete, the user and system collaborate in creating the prototype through step-by-step implementation, where the system breaks down the implementation plan into steps (see Fig 3- F). Because project specifications are often complex and detailed, attempting to complete the application with a one-shot approach often results in key elements being missed, leading to non-functional prototypes. With missing functionalities, users can spend unnecessary time debugging and adding features to align with the initial spec. Breaking down the spec into manageable steps ensures that details are not missing

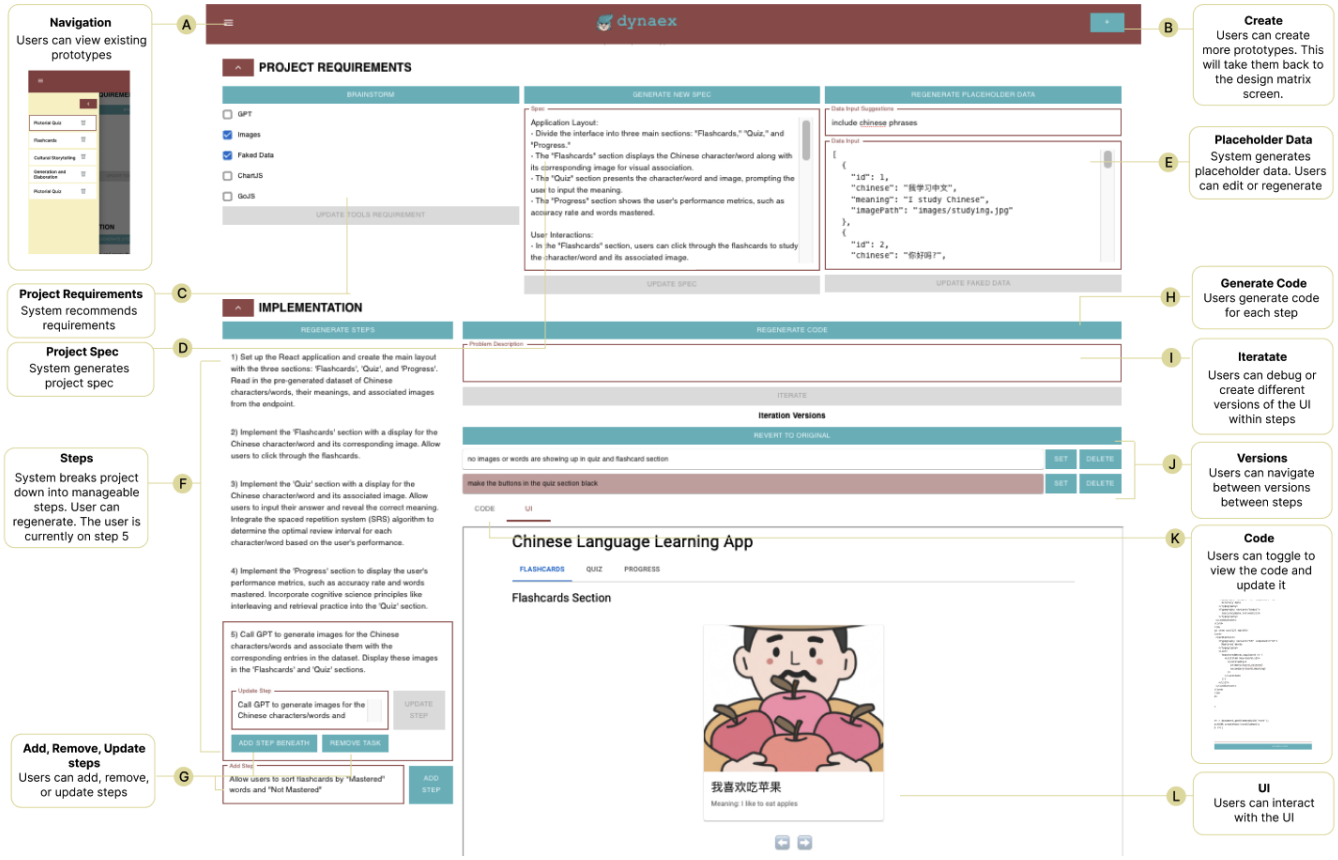

Figure 3: DynEx's Implementation User Interface: Users can navigate between existing prototypes (A) and create new prototypes (B). Users are suggested and can select from a list of potential project requirements (C). Users are able to edit the project specification (D) and view/modify placeholder data if it is required (E). DynEx breaks down the project specification into implementation steps (F) which can be edited, added, and removed by users (G). Users generate code (H) step-by-step. Users can iterate via natural language (I), toggle between versions (J), and view generated code (K) and interact with UIs (L).

in the final prototype. Furthermore, our iterative step-by-step approach saves the code at each stage, providing a natural form of version control.

3.2.1 Project Requirements. Before implementation, the system identifies prerequisite project requirements and generates a (spec) (see Fig 3- C, D, E). The project requirements detail whether or not the prototype requires GPT, images, placeholder data, or supported external libraries. The system brainstorms these requirements using the Design Matrix as context. Users can manually update and change the project requirements if needed - for example, if the application is a book recommendation app, and DynEx did not recommend the images as a requirement, the user can add that requirement themselves if they so please. Using the project requirements and Design Matrix as context, the system then generates a project specification to translate the abstract ideas across each dimension into a technical implementation.

Self-invoked Multi-Modal LLMs. Our project requirements include the ability to call GPT and generate images. To do this, we introduce a technique to self-invoke multi-modal LLMs, enabling the creation of a diverse suite of applications. This allows our code to generate dynamic data, create images, and even return code through LLMs. In software development, it is common to rely on third-party application programming interfaces (APIs) to enhance functionality. For example, travel websites can call the Google Flights API for flight information, and restaurant recommendation applications can call the Yelp API for restaurant data. Generative AI APIs are among the most powerful and versatile, excelling at handling a wide variety of tasks. By self-invoking multi-modal LLMs, we can build applications that are otherwise not supported by static data; for example, book recommendation applications that require dynamic data from LLMs, wardrobe visualization applications that can call image models, and note-organizational applications that can use LLMs to categorize notes and sensemake. LLMs are a powerful tool to self-reference when generating code for exploratory programming. It allows us not only streamline the development process and rapidly prototype applications, but also create more dynamic and functional UIs.

Dynex self-invokes GPT-4 and DALL-E 2 to enable dynamic and flexible content creation based on the project requirements. We use few-shot examples to support this technique; if GPT or Images is selected as a requirement (see Fig 3- C), we provide different code examples of calling GPT-4 or DALL-E 2 in the back-end prompt to guide the self-invocation. If these requirements are not needed, we omit those examples. For our example to learn Chinese, "images" is selected – DynEx will self-invoke DALL-E 2 to generate images for the prototype.

External Libraries. DynEx further enhances functionality with pre-approved libraries like Chart.js, an open source javascript library used for creating interactive and visually appealing charts on websites, and GoJs, as javascript library for creating interactive diagrams such as flow charts, mind maps, and process diagrams. A common issue with LLM code-generation tools is their difficulty in handling external libraries and packages. OpenHands can get stuck in an loop installing unusable packages, while tools like Claude Artifact fail when trying to import non-existent component libraries. These limitations pose significant challenges for generating reliable code with LLMs. To prevent this, we restrict our system to use only the MUI React Library and third-party API calls outside of OpenAI. However, this restriction can limit the complexity of generated applications, so we allow the use of Chart.js and GoJS for creating charts and diagrams. When our system recommends these libraries as requirements, we use few-shot code examples to ensure smooth integration. We can expand the list of pre-approved libraries this way to ensure safe and reliable code generation. For our example, neither Chart.js or GoJS is identified as a project requirement.

Project Specification. DynEx generates a project specification (spec) (see Fig 3- D), a detailed overview of the prototype idea which connects the concepts from the Design Matrix to flesh out the application layout, user interaction, and logic of the app. The spec transforms abstract ideas into actionable implementation instructions, ensuring the functionality of the app is well-aligned with the design; it provides a clear roadmap for development. Although there is potential overlap between the spec and Grounding level in the Design Matrix, the project specification focuses more on the technical requirements for the project. If placeholder data is needed, the spec helps define the schema. If GPT must be self-invoked, it details what the code should prompt for and the expected response. Ultimately, the spec drives the implementation plan; DynEx breaks the spec down into steps to iteratively implement on later on in the workflow.

In this example, the system determines that placeholder data and images are necessary components for the UI to be realized. The spec details that the system must generate placeholder data with fields that include Chinese words and phrases, images, and IDs. It also details that images must be returned for each quiz question to assist with visual learning.

Synthetic Placeholder Data. DynEx also enables the generation of synthetic placeholder data (see Fig 3- E). By using the spec and the Design Matrix as contextual inputs, it creates realistic placeholder data to serve as the foundation for building data-driven prototypes. We leverage few-shot examples with LLMs to guide the generation of relevant and appropriate placeholder data. The user can regenerate, approve, or modify the outputs. In our example, the user generates placeholder data, providing the prototype with a solid base of Chinese phrases to practice with, making this step essential for effective app development.

3.2.2 Step-by-step Implementation. To implement the prototype, we employ a dynamic, iterative approach to development that allows for human direction at every step, which can be seen in Fig 4. The system breaks down the spec into steps, where each step is the next-smallest testable iteration of the previous step. In our example, the step list is broken down to:

(1) *"Set up the React application and create the main layout with the three sections: 'Flashcards', 'Quiz', and 'Progress'"*
(2) *"Implement the 'Flashcards' section"*
(3) *"Implement the 'Quiz' section, utilizing the spaced repetition algorithm"*
(4) *"Implement the 'Progress' section"*
(5) *"Call GPT to generate images for the Chinese characters"*

The user can regenerate, approve, or modify this task list.

The system next executes steps sequentially. In our example, the system implements (1) *"Create the HTML structure"*. The system generates the initial code, and the UI is rendered for the user to interact with. If there are failures, the user can debug by iteratively prompting the system in the right direction to regenerate code (see Fig 4 - I), redoing the step (see Fig 4 - H), or debugging the code by hand (see Fig 4 - K). Once the user confirms that it works, they can proceed to (2) *"Implement the 'Flashcards section'."* The user does not modify the step and clicks the button to generate the code. Once the user tests and approves step 2, they can move on to (3) *"Implement the 'Quiz' section."*

Debugging and Iteration within Steps. At (3) *"Implement the 'Quiz' section,"* there's an error with the "submit" button in the 'Quiz' section. Instead of displaying the next quiz question, it displays both the next quiz question and the next quiz question's answer. To fix this, the user can make use of the "Iterate" box to debug the program by explaining the bug through the prompt (see Fig 4 - I). The prompts the system to fix this bug: "After I click submit, it moves on to the next question but also shows the answer of the next question," and clicks "Iterate". After generating the code, the debugged UI and code is shown on the system. Users can iterate as many times as they want and switch between versions until they are happy with the final step output(see Fig 4 - J).

Feature Adding. The user tests and approves step 3, and moves on to steps (4) *"Implement the 'Progress' section"* and (5) *"Call GPT to generate images."* Thus, the user has completed the original implementation plan. Dynex enables users to add more features as they test prototypes. After using the app and testing it out, the user decides they would like another feature: in the flashcards section where they practice the words, they would like a visual indicator to show which words they have mastered, and which words they are not familiar with. The user adds a step 7 to do so (see Fig 4 - G).

Version Control. By breaking our project into distinct steps, where users can update, remove, and add features, we enable version control. Each step is saved, creating a clear version history that captures

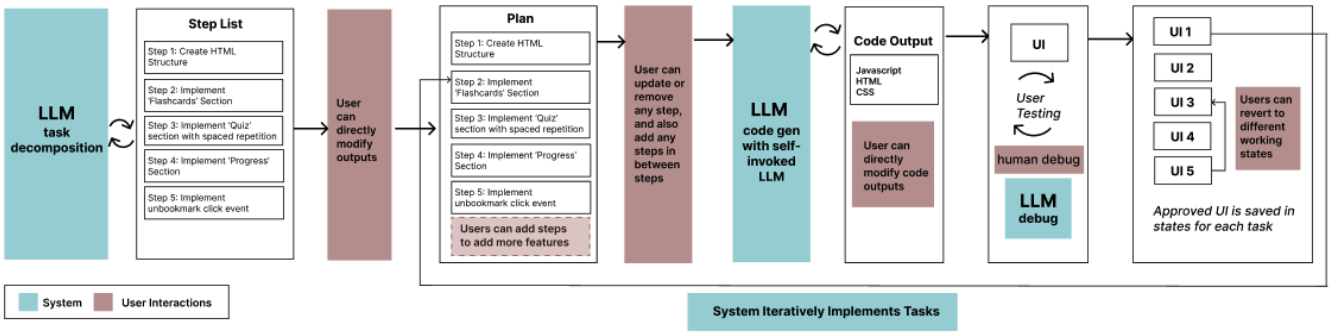

Figure 4: DynEx User Process: The system first decomposes the project into steps that are modifiable by the user. The user iteratively implements each step. The user can add, remove, or update steps at any point. The backend LLM then generates the UI code – the user can also interact with the UI . The user can directly modify the code output, debug, and revert to previous versions. The user iterates through steps until the final prototype is complete

the most advanced working prototype while offering the flexibility to iterate between versions. This means that users can easily revert to prior working states if they want to reconsider a feature, without worrying if leftover changes will affect the process.

Since LLM code-generation can sometimes produce faulty or unusable code, this stepwise approach serves as a safety net where users can always fall back on the last successful version, ensuring that they always have a stable point to return to. While this is not a main contribution of our system, it's essential to support version control in exploratory programming to provide structure and safety in an unpredictable process, protect users from irreversible errors, and support smooth iteration.

3.3 Creating Multiple Prototypes

Once a prototype is created, users can create additional prototypes by revisiting the Design Matrix, tweaking parameters, exploring variations, and comparing the results (see Fig 4 - B). The different prototypes are displayed in the sidebar of DynEx (see Fig 4 - A), allowing users to easily revisit any prototype, pick up where they left off, and continue iterating. Users can modify entries in the Design Matrix and re-implement new designs to generate a diverse set of prototypes. This is a consistent method with exploratory programming, empowering users to explore multiple solutions in parallel and refine their designs through hands-on iteration.

4 Evaluation

To evaluate DynEx as a system for exploratory programming, we conducted a user study that focused on the following research questions:

- **RQ1: Divergence** - To what extent does DynEx enable divergent exploration within a problem space?
- **RQ2: Convergence** - To what extent does DynEx allow users to better develop their idea?
- **RQ3: Implementation** - To what extent does DynEx enable the code to realize a complex idea?
- **RQ4: Overall** - To what extent does DynEx allow for a better prototyping experience?

4.1 Methodology

4.1.1 Participants. Our evaluation was conducted through a qualitative study of 10 programmers (6 male, 4 female); there were 6 industry software developers and 4 CS students, all with many years of technical experience. They were recruited through a snowball sampling of local university students and alumni. Information regarding the participants age, role, and years of technical exposure are included in Table 1. Participants were paid $20 per hour for their time, and the study was conducted with each participant for approximately 90 minutes.

4.1.2 Task. Participants were asked to name a problem that they would like to explore a solution for. The task was to explore concepts and prototypes that would help solve their problem. They were asked to produce a MVP per system. Participants had a broad range of ideas, from recommendation applications for movies, clothing, and restaurants, to a friend-activity sharing app, to a rap ghost-writing assistant (see Fig 6). They tested two systems: DynEx and Claude Artifact.

4.1.3 Claude Artifact. We compared DynEx against Claude Artifact, an industry standard tool, as the baseline. Claude Artifact is a chatbot, where every prompt elicits a UI and non-editable code. The chat screen is on the left, and the Artifact to interact with the UI is on the right. The user can prompt the chatbot to brainstorm more ideas or specify more features – an example can be seen in Figure 5.

Because Claude Artifact can generate functioning code from short prompts, it is a very powerful tool for prototyping. Claude Artifact is also widely accessible and creates visually-appealing UIs. Although it is limited to 4096 output tokens (approximately 450 lines of code), Claude Artifact is a strong baseline because it still produces rich prototypes and stores long chat histories that track how the user further developed their application from the initial idea.

We considered other tools as the baseline, including OpenHands and GPTPilot. We found OpenHands to be less reliable than Claude Artifact in producing functional prototypes. In our exploration with

ID	Age	Gender	Role	Years of Technical Exposure
1	20	M	Undergraduate Computer Science Student	3
2	25	F	Software Engineer at Large Company	7
3	24	F	Software Engineer at Large Company	6
4	22	F	Software Engineer at Large Company	5
5	25	M	Software Engineer at Large Company	7
6	25	M	Software Engineer at Large Company	7
7	21	F	Master's Computer Science Student	5
8	24	M	Doctoral Computer Science Student	6
9	24	M	Doctoral Computer Science Student	6
10	26	M	Software Engineer at Start Up	5

Table 1: Demographics of user study participants. Technical Exposure refers to the number of years since the participant first programmed.

GPTPilot, we found its user interactions limited to technical specifications. It did not allow for sufficient design space exploration, which is largely emphasized in DynEx. Since the user has the ability to engage with design space exploration through the flexibility of the chatbot interface, we decided Claude Artifact was the most appropriate baseline for our study.

4.2 Design Exploration through the Design Matrix

4.2.1 Procedure. An experimenter first explained the concepts of DynEx through a slide deck that introduced design exploration and exploratory programming, focusing on user-centric design principles and LLM code-generation. The experimenter then gave a demonstration of both systems. For Claude Artifact, the experimenter demonstrated how to prompt it to create a UI and add features through the chatbot. For DynEx, the experimenter demonstrated how to use the Design Matrix and iteratively implement, debug, and add features to their prototype. After the demonstrations for each system, participants were given 30 minutes to create a prototype using both DynEx and Claude Artifact. Half of the participants started prototyping with Claude Artifact, and the other half started with DynEx. After the conclusion of the two tasks, an experimenter conducted a semi-structured interview to understand participant experiences.

4.2.2 Ratings and Interviews. After using both systems, participants were asked to rate their experience across a 1 (bad) to 7 (good) scale for 10 questions. The first 4 questions were directed towards the exploratory programming experience. The last 6 questions measured the participant workload while using each system, using the NASA Task Load Index (NASA-TLX) [14]. NASA-TLX can assess the overall success of a system because it provides a comprehensive measure of cognitive workload, which directly influences user performance and satisfaction. Users were also interviewed afterwards about their experience with both systems and their thoughts on the prototypes generated.

4.3 Results

4.3.1 RQ1: Divergence - To what extent does DynEx enable divergent exploration within a problem space? Our evaluation showed that participants found it much easier to explore different design ideas when using DynEx as compared to a baseline Claude Artifact. Across 10 participants, DynEx was rated significantly higher than Claude Artifact in enabling design exploration (average scores of 6.1/7 and 4.0/7, respectively; see Table 2 or Figure 7). 8/10 participants rated DynEx at least a 6/7 in regards to this metric. A paired t-test shows that difference in ratings of the two systems is statistically significant at the p = 0.05 level (see Table 2).

Participants reported that DynEx's system-led brainstorming was more effective in exploring different possible solutions. All participants noted that the system articulated required components of the application they recognized, or interesting possibilities they had not considered. For example, P8 stated, *"I didn't start with that many ideas to begin with on my own. [Without DynEx] I really wouldn't have thought of these steps that fast. It was predicting more things than I had imagined."* All participants similarly expressed positive sentiment about being presented with ideas while using DynEx, feeling that using Claude Artifact required significantly more mental exertion to come up with such ideas themselves. **From these results, we conclude that DyNex enables users to explore a broader range of solutions to their problem.**

4.3.2 RQ2: Convergence - To what extent does DynEx allow users to better develop their idea? We found that participants were able to better develop their initial idea while using DynEx as compared to a Claude Artifact baseline. Across 10 participants, DynEx was given an average score of 5.9/7 for convergent thinking, compared to an average score of only 3.4/7 for Claude Artifact (see see Table 2 or Figure 7). 7/10 participants rated DynEx at least a 6/7 for convergent thinking. A paired t-test shows that the difference in ratings of the two systems is statistically significant at the p = 0.05 level (see Table 2).

Participants indicated that they were able to sufficiently build off of their own ideas using DynEx. P9 stated that DynEx allowed

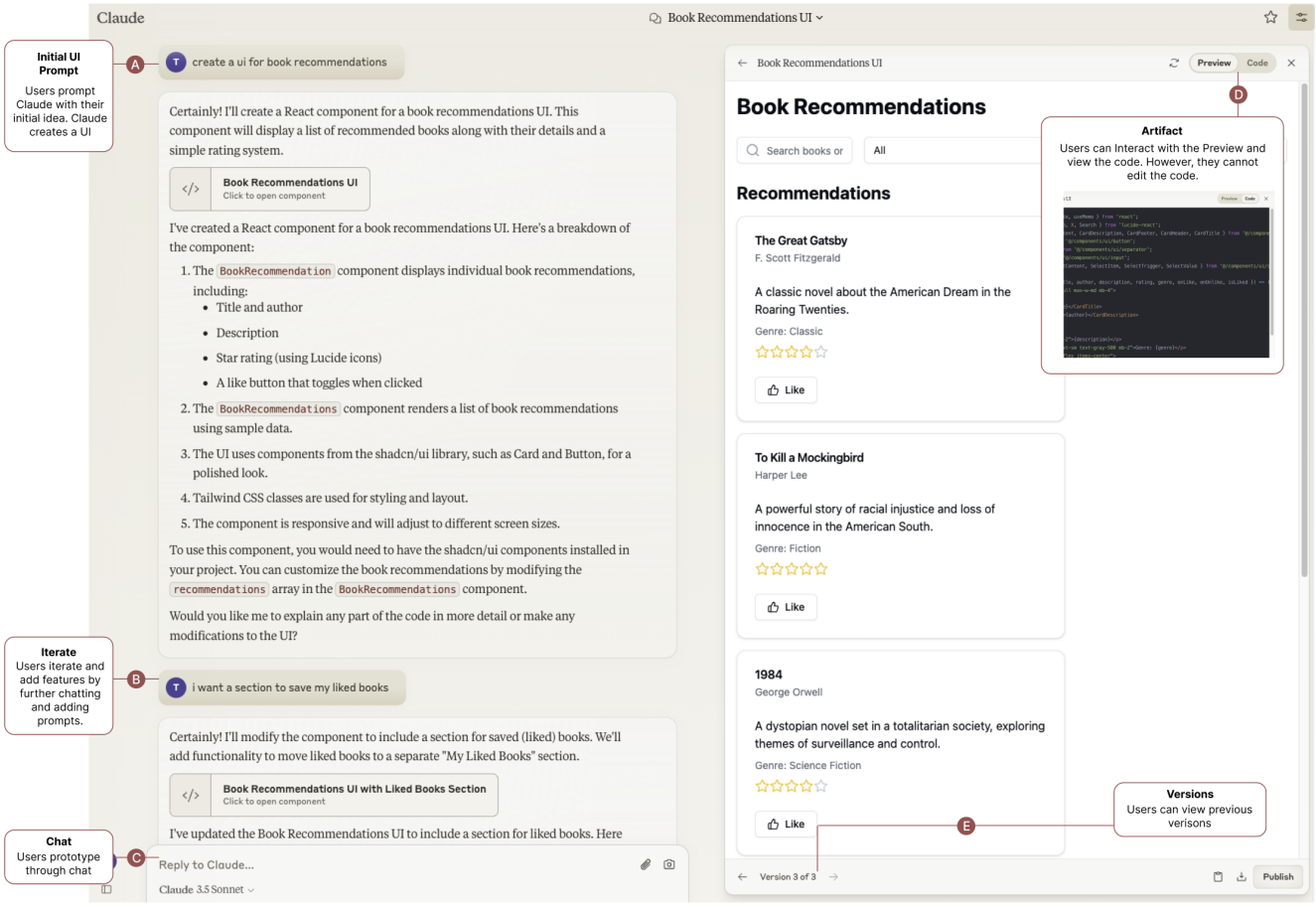

Figure 5: Claude Artifact Interface: Claude Artifact has a chat interface, where users can prompt the chat and have code returned instantly. Users can view the code and the UI on the artifact to the right; users cannot edit the code. Users can iteratively append features to the initial prototype through chat.

them to *"incorporate [their] initial idea and helped [them] refine it."* Participants also reported that using DynEx's Design Matrix allowed them to better specify the problem that they were trying to solve, specifically noting that the Person dimension was helpful to inform the remaining entries of the matrix. Participants also felt that they were better able to explore the solution space with DynEx as compared to Claude Artifact. Overall, DynEx was as very helpful during the design process; P8 stated that they *"didn't think many [questions were] left unanswered."* **From these results, we conclude that DynEx is more successful at developing users ideas.**

4.3.3 RQ3: Implementation - To what extent does DynEx enable the code to realize a complex idea? Our evaluation demonstrated that participants were able to create more feature-rich and complex solutions using DynEx as compared to Claude Artifact. The 10 participants scored both systems similarly in terms of code-generation realizing their ideas (see Table 2 or Figure 7), but there was a major difference in participant scores regarding the complexity, which we define as the feature-richness of applications, that were created. Participants scored DynEx (average of 4.9/7) higher than Claude Artifact (average of 3.7/7) in terms of the complexity of the application produced (see Table 2 or Figure 7), with several describing their DynEx prototypes as being *"more feature-rich"* than their Claude Artifact counterparts. 8/10 participants scored DynEx prototypes as being at least a 5/7 in terms of complexity, whereas only 3/10 participants scored their Claude Artifact prototypes above 5/7. A paired t-test shows that the difference in ratings of the two systems in terms of complexity is statistically significant at the $p = 0.05$ level (see Table 2). **From these results, we conclude that DynEx is able to create more complex and feature-rich applications.**

4.3.4 RQ4: Overall - To what extent does DynEx allow for a better prototyping experience? Overall, we found that DynEx was more successful for exploratory programming compared to Clauade Artifact. In terms of success at accomplishing the task at hand (NASA-TLX performance), participants on average scored DynEx 5.9/7

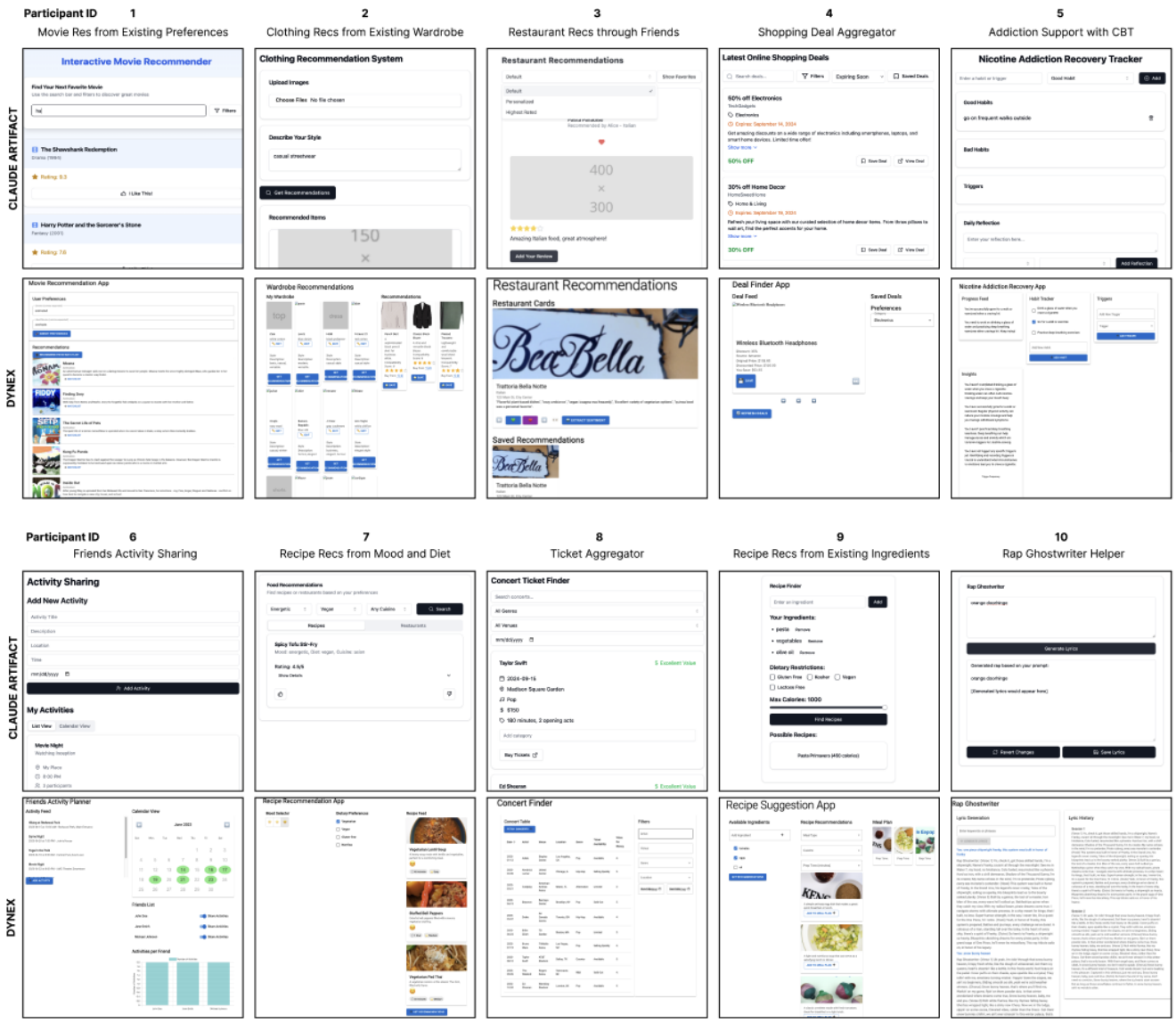

Figure 6: Participant Outputs: UI prototypes created by all 10 participants using both Claude Artifact and DynEx. These UI prototypes spanned a diverse range of use cases from addiction support with CBT to a rap ghostwriter helper. DynEx allowed participants ground prototypes around the real-world constraints of a problem such as the Person, Approach, and Interaction.

compared to only 4.2/7 for Claude Artifact (see Table 3). 8/10 participants rated DynEx a 6/7 or higher in terms of NASA-TLX performance. This difference in rating was statistically significant at the p = 0.05 level (see Table 3).

The difference in ratings between DynEx and Claude Artifact for the remaining NASA-TLX metrics (mental demand, physical demand, temporal demand, effort, and frustration) were statistically insignificant (see Table 3). However, there were still interesting findings for the mental effort and demand metrics. The NASA-TLX score for both had a large range of responses; many participants found that DynEx gave them freedom to think conceptually, but also required them to read and wait more as compared to Claude Artifact. However, while using Claude Artifact, users expressed that it required more mental exertion to brainstorm ideas and guide the prototypes development direction. Hence, depending upon user's preferences (i.e., do they prefer reading LLM-generations or writing prompts themselves), scores for mental exertion may vary. This same discrepancy likely explains the lack of statistical significance in the difference in ratings for the NASA-TLX effort category. **From these results, we conclude that DynEx more successfully allowed users to create a final application as compared to the Claude Artifact baseline.**

Exploratory Programming	Claude Artifact	DynEx	p-value	t-statistic
Divergent Thinking: To what extent were you able to explore different conceptual options using *system X*?	4.2	6.1	**0.0066**	-3.52
Convergent Thinking: To what degree were you able to better develop/specify your idea using *system X*?	3.4	5.9	**0.00015**	-6.23
Idea Realization: To what degree was the code able to realize your idea when using *system X*?	5.2	5.5	0.678	-0.43
Application Complexity: How complex is the prototype produced by *system X*?	3.7	4.9	**0.024**	-2.71

Table 2: Average Participant Score for Exploratory Programming Metrics Between Claude Artifact and DynEx. For all questions, a score of 7 is best (i.e., for all comparisons, a higher number is better). The results from paired t-tests are also listed; statistically significant p-values (p < 0.05) are in bold.

Mental Workload/NASA-TLX	Claude Artifact	DynEx	p-value	t-statistic
Mental Demand: How mentally demanding was the task when using *system X*?	3.2	4	0.280	1.15
Physical Demand: How physically demanding was the task when using *system X*?	5.7	5.7	1.0	0.0
Temporal Demand: How hurried or rushed was the pace of the task when using *system X*?	4.7	5.4	0.242	1.25
Performance: How successful were you in accomplishing what you were asked to do when using *system X*?	4.2	5.9	**0.003**	-4.02
Effort: How hard did you have to work to accomplish your level of performance when using *system X*?	4	4.8	0.196	1.39
Frustration: How insecure, discouraged, irritated, stressed, and annoyed were you when using *system X*?	4.6	4.8	0.785	0.28

Table 3: Average Participant Score for Mental Workload/NASA-TLX Metrics for Claude Artifact and DynEx. For all questions, a score of 7 is best (i.e., for all comparisons, a higher number is better). This includes questions such as mental/physical demand - a 7 indicates that there was low exertion. The results from paired t-tests are also listed; statistically significant p-values (p < 0.05) are in bold.

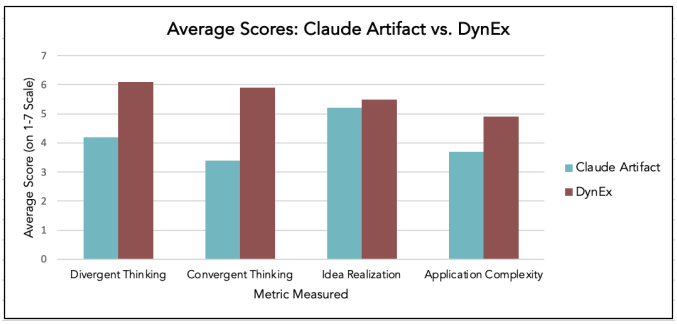

Figure 7: Average Scores for Claude Artifact and DynEx for exploratory programming metrics. The difference in averages is statistically significant for Divergent Thinking, Convergent Thinking, and Application Complexity.

4.4 General Qualitative Findings

4.4.1 DynEx's Design Matrix inspired new solutions. DynEx's Design Matrix enabled participants to consider ideas they had not thought of, broadening their initial concept and allowing them to adequately explore the design space. P3 prototyped an app that recommended restaurants based on friends reviews. She noted: *"It was really cool that it gave me ideas – otherwise I would have gone with what I had imagined ...[initially, which was not as good]"*. She had pictured a typical, feed-like-layout, but through DynEx's exploration, ended up prototyping a card-swipe interaction that she enjoyed more.

P5 experienced something similar, stating that *"For every [entry in the matrix], it included [ideas] that I wanted, but also considered options that I hadn't considered but wanted to build...it did a good job of articulating ideas that I had in my mind, touching on tangential pieces that I might be interested in but didn't know how to bring together."* P5 created a addiction recovery application, and was particularly intrigued by using cognitive behavioral theory (CBT) principles brainstormed in the Approach dimension. He ultimately

created an app that allowed him to journal triggers, relapses, and good days, utilizing CBT help identify unhelpful thoughts.

P8 created a prototype to aggregate concert tickets based on value-for-money – he had been thinking about this idea for the past two years, but the Design Matrix was able to help him make progress with new insights. He stated,*"even though I kind of knew what I wanted to do, I really wouldn't have thought of these steps so fast. Stuff like: 'do you want a centralized database?'... [for even] simple things like that, I would have to think for a much longer time if I wanted to build it myself... it saved me from spending a lot of time on unimportant things... and gave me conceptual freedom."*.

On the contrary, Claude Artifact did not facilitate the exploration of the design space, but rather, implemented prototypes exactly as the user prompted. P2 noted that *"with DynEx, there's a lot more time put into the [end-user of the prototype] and [a lot more] feature-oriented thoughts...[with Claude], I didn't know what to [prompt] at times... since there were no recommendations."* Additionally, P4 stated that when using Claude Artifact, *"[he] was just thinking on [his] own... it did not come up with any of the ideas, [he] came up with all the ideas"*. The Design Matrix gave users a better starting foundation; P9 stated that DynEx allowed him to *"incorporate [his] initial idea and refine it... the brainstorming helped [him] think through rough points [he] had missed out on"*. Exploring the design space prevented users from design fixation, allowed them to ideate on new solutions, and challenged their existing thinking.

4.4.2 DynEx's prototypes are more feature-rich and intuitive. Users were also pleasantly surprised by DynEx's ability to realize more intuitive and realistic applications. DynEx was able to generate more relevant placeholder data and emulate a true user experience. P2 noted that *"having the faked data really helps with understanding what is going on with the system and interacting with it"*, compared to Claude Artifact's *"very simple"* prototype with limited placeholder data. P3, who created the restaurant recommendation app, stated that she could not emulate a user experience from her prototype with Claude Artifact because *"[her] data could not mimic it... [it didn't even] have reviews"*; in comparison, DynEx's placeholder data *"helped a lot [for her to envision] the UI properly"*.

DynEx's ability to consider many small details that worked together cohesively also helped realize more thoughtful and intuitive prototypes. P6 had a vague application idea when initially prototyping – a friend-activity sharing app where friends could join other friend's plans. He eventually settled on a calendar-like interface, but was still surprised by the comprehensive feature-set the prototype provided, stating, *"it suggested many features to answer the foundational set of questions that were trying to be answered, such as how to match people with friends, how to visualize the friend-sharing, joining friend-sharing experiences... it created a pretty robust social ecosystem surrounding the calendar experience."* He felt as though the final prototype *"incorporated really different, very distinct UI features that weren't very connected to each other [at face value] very well... it solved the problems it [set out to solve]."* He indicated interest in using this initial prototype as a starting point for a real, production application for himself to build off of.

P8 also noted the intuitive features in his concert aggregator prototype. With DynEx, the UI was formatted like a table, with important information surfaced – the main user interaction prioritized searching for valuable concert tickets with low cognitive load. In contrast, the UI for Claude Artifact had a feed view, with lots of information about each concert and ticket, which P8 disliked after interacting with the prototype. P8 greatly preferred DynEx's prototype, stating: *"It was intuitive. It was feature-rich. It had all the important features, like sorting by columns and sorting by genre, dates, value-for-money etc...I liked the conciseness of the information... contrary to [Claude Artifact's] prototype which was... not-fit for this use case."*. Ultimately, users were satisfied with the cohesion between features in DynEx-created prototypes.

4.4.3 Claude's immediate feedback loop is engaging. Users viewed Claude Artifact's responsiveness favorably; they were able to quickly generate a UI based on limited prompts. P4 liked that Claude Artifact *"showed the prototype instantly"*. P5 noted: *"Claude is a chat, which is effortless... because it didn't force me to explore more, it didn't feel mentally demanding."* Similarly, P9 stated that Claude Artifact's *"translation from prompt to output to previewing code is very clear and fast."* Users disliked Dynex's slower code-generation and the dense-text they were required to read through when traversing through the Design Matrix. P9 said that DynEx required *"a bit more mental effort because you had to think through...navigate... and read through the matrix"*. Many users also commented on the speed of the code generation; P1 stated that DynEx was *"very slow"* compared to Claude Artifact. Almost all participants appreciated the speed of Claude Artifact compared to DynEx, emphasizing the importance of an immediate feedback loop.

However, participants did also note that this was a drawback in terms of enabling a thorough design space exploration – because of Claude Artifact's inherently linear workflow, users only appended features to their initial prototype, as opposed to trying out different ideas. P4 noted: *"[Claude Artifact] just did what I told it to do... [the prototype] was what I expected"*. P8 stated: *"I couldn't even brainstorm the fact that it could brainstorm,"* reflecting a broader trend among participant usage patterns with Claude Artifact; participants did not spend much time ideating with Claude Artifact because it was not intuitive on how they could do so.

4.4.4 DynEx bridges the gap between design and engineering. Participants appreciated the broader context and conceptualization provided by the Design Matrix. Participants in our study had significant technical experience but limited design experience, with 6/10 currently working as software engineers in industry. P2 stated: *"When it asks who is the impacted user, [it is like] what our design person always asks."* P8 said: *"I appreciated that [DynEx] walked me through the actual entire design experience, similar to how a product manager thinks about a business question - thinking about who's the user? What are the user stories surrounding the users? And then mapping those user stories to actual features."* They appreciated the contextual understanding about the design they gained through traversing the Design Matrix.

5 Discussion

DynEx successfully enabled exploratory programming, facilitating the transformation of abstract ideas into concrete prototypes. We discuss how LLMs can significantly enhance and accelerate the

process of exploratory programming. We additionally consider the implications of unstructured and structured approaches towards design exploration. Finally, we discuss the potential of the Design Matrix acting as a liaison between designers and engineers.

5.1 LLM Techniques to Enhance Exploratory Programming

One of the biggest challenges when using LLMs for design exploration is guiding users towards adequate problem specifications. DynEx was able to address this by providing users with a framework for structured exploration of a design space based on specific dimensions (Person, Approach, and Interaction). In many instances, DynEx offered suggestions users intuitively understood but could not fully articulate, supporting users with recognition over recall. Exploring dimensions of the design space with DynEx also broadened the scope of the exploration process. Our dimensions provided natural parameters that users could repeatedly change to create variations of prototypes. Ultimately, the true power of exploration through dimensions lies in helping users specify their problems, clarifying their design goals, and broadening their landscape of solutions.

LLM code-generation abilities can transform the exploratory programming process. Exploratory programming systems must consider the ease or difficulty of exploration when prototyping. With LLMs, users can prototype UIs rapidly without compromising the complexity of the application through self-invoking multi-modal LLMs. While prototypes should be simple, they need a minimum level of complexity in order to properly mimic the user experience. The self-invocation of generative AI allowed users to better simulate a real application through images, placeholder data, and dynamic data generation. It is a powerful idea for LLMs to leverage their own advanced capabilities to create more functional applications – without it, recommendation systems wouldn't be able to recommend anything and applications with a visual component of the experience would have no images to flight-test the experience.

To enable users to explore the design space even more thoroughly, we can incorporate dimensions past the Person, Approach, and Interaction, as well as levels of specificity beyond Idea and Grounding. For example, a new dimension that could be considered is multiple stakeholders – a marketplace application must have functionalities for both buyers and sellers. Another potential dimension is existing solutions – if designing an application for a problem for which other solutions already exist, users should be able to identify deficiencies in existing solutions to inform the development of their own prototypes. Adding dimensions leads to more thoughtful application designs that take into account more aspects of the problem. Additionally, we could add levels of specificity outside of the Idea and Grounding levels, such as a row for mock-ups that provide a visual of the dimension. Adding levels of specificity to further crystallize concepts can result in the creation of a more-detailed and cohesive application.

5.2 Structured vs Unstructured Design Exploration

While users felt supported by DynEx's structured design space exploration, they also appreciated aspects of Claude's chat-interface – namely, that it provided a more immediate feedback loop. Claude Artifact's interactions are simple and direct – users could type an input and almost instantly receive a code output. This responsiveness engaged users, motivating them to iterate more quickly on ideas. Chatbot systems are very common; people keep making them, and users are very familiar with them. They invite open-ended input, allowing users be highly specific or intentionally vague in their prompts. However, just because it is common, does not mean it is the best paradigm. Comparing it effectiveness to a structured UI, such as DynEx, is also important.

We found that users do not really use Claude Artifact to explore a problem space, despite the fact that they could prompt the chat to help them explore. Most participants instead worked through a linear process, appending features to Claude Artifact's initial design. Linear processes are known to be problematic because they do not facilitate an exploration of the broader design space, limiting the complexity and intuitive way that features can work together in the application. In fact, all users were unaware or did not even think of utilizing Claude Artifact for exploration outside of feature-addition purposes.

It remains an open question as to exactly how structured design space exploration should be without compromising the implementation process. DynEx's Design Matrix required users to explore the problem space before implementation, but users may not always be ready to engage with the detailed considerations that structured exploration demands. Future exploratory programming tools should consider ways to combine the immediate feedback and flexible prompting of Claude Artifact with the comprehensive design exploration of DynEx. Perhaps a user could begin with with unstructured exploration, and once sufficiently inspired, transition to a more structured approach to tackle more complex solutions. Designing prototyping tools that can strike a balance between ease and structure could encourage more individuals to explore unrealized ideas, foster innovation, and solve real-world problems.

5.3 Bridging the gap between design and programming

DynEx brings together design and programming for the user. Traditionally, designers and engineers are siloed during software development - designers design and engineers implement, with product managers attempting to bridge the gap between the two. However, the design landscape is always changing, and context is bound to be lost – even a product manager may not be up to date on the latest changes. Addressing this disconnect between design and engineering is challenging but likely useful; our user study with individuals demonstrated the value in contextualizing implementations in design. However, in the working world, designers and engineers work in teams. Designers must continuously update engineers through shared Figma files as designs change. LLMs offer a unique opportunity to provide a continuous feedback loop between teams of designers and engineers and reduce this friction. Existing LLM-powered tools for established codebases, such as CoPilot and Gemini, only support developers in implementation. DynEx supported exploratory programming with design exploration for individuals with successful results; the idea of incorporating design

context into these LLM-powered systems could extend to tools for software development workflow on teams.

6 Limitations and Future Work

Our user study was limited to ten participants, all of whom had computer science backgrounds and many years of technical experience. This may not be representative of the broader population that could benefit from an exploratory programming system – namely, anyone with an idea to prototype, regardless of their past technical exposure. Future studies should expand the demographics of participants to best inform the development of a system for users of varying technical backgrounds. Our study was also limited in the amount of time participants had to prototype. They had 30 minutes with each system – with additional time, participants may have created a variety of prototypes and compared them with one another to emulate a richer prototyping experience. Future studies should examine how users interact with the system when given more time.

Our user study also only used Claude Artifact as a baseline. Although we conducted a brief exploration which suggested Claude Artifact as being the most suitable as a baseline compared to other existing LLM-based tools, future work could involve comparisons between more systems in order to identify specific components or design-choices that are effective in supporting exploratory programming. Furthermore, while users were given an introduction to Claude Artifact, they were still quite inexperienced; experienced users might use Claude Artifact as a tool for prototyping differently than novices.

Our system uses Claude 3.5 Sonnet for coding, and while very powerful, it introduced certain limitations. Most notably, Claude has a token cap of 4096 tokens, where API calls can only return approximately 450 lines of code. 450 lines of code is enough to prototype most applications; however, users were limited in complexity when they attempted to add features. Additionally, users expressed dissatisfaction with DynEx's speed, noting that Claude Artifact offered a faster feedback loop. API responses from LLMs often return undefined code, requiring extra steps for cleaning and debugging the code in our system, which slowed down the prototyping process. As LLMs improve, these "clean-up" steps will also become unnecessary, providing a faster experience. Finally, because our system is built on top of LLM capabilities, which are far from perfect, they could return sometimes buggy or broken code. Users had to either regenerate or manually debug to handle these failures. We expect these issues to improve as LLMs themselves do. As new foundational models are released, they should continually be inserted and experimented with within the system to measure effects on performance.

While traversing through the Design Matrix while using DynEx, users expressed a desire for simpler and more readable language. Future work can integrate LLM-summarization capabilities or other NLP techniques to improve clarity, succinctness, and cognitive load. In addition, DynEx currently does not allow users to continuously brainstorm additional features during implementation. It would be beneficial to help users identify what other features to explore once the initial prototype is complete, instead of the user being left to their own devices in improving the initial prototype.

Code-generation abilities of LLMs will undoubtedly improve and accelerate exploratory programming; blending design exploration and implementation must also be aligned with these developments. Our work explores the role of the Design Matrix in understanding the problem space for exploratory programming. Future work should continue investigating other approaches to how LLMs can effectively bridge the gap between design and implementation and enhance the exploratory programming process.

7 Conclusion

The advancement of LLMs poses a unique opportunity for exploratory programming. LLMs can generate code rapidly and also help users enhance and refine their problem space. In this paper, we presented DynEx, a system to accelerate exploratory programming through a structured, LLM-guided Design Matrix and implementation through iterative LLM code-generation. We also introduce self-invoking multi-modal LLMs as a technique for LLM-based exploratory programming, allowing for faster creation of diverse and functional prototypes. Through our user evaluation with 10 technical experts, we found that DynEx's Design Matrix allowed to users to explore, refine, and realize feature-rich applications suitable for prototyping. We conclude by discussing how systems like DynEx can help bridge the gap between design and implementation, empowering a broad range of individuals to bring their ideas to life.

References

[1] Leyla Alipour, Mohsen Faizi, Alireza Moradi, and Gholamreza Akrami. 2017. A Review of Design Fixation: Research Directions and Key Factors. *International Journal of Design Creativity and Innovation* 6, 1-2 (2017), 22–35. https://doi.org/10.1080/21650349.2017.1320232

[2] Anthropic. 2024. *Claude 3.5 and the Art of the Sonnet.* https://www.anthropic.com/news/claude-3-5-sonnet Accessed: 2024-09-11.

[3] Ofer Arazy, Oded Nov, and Nanda Kumar. 2015. Personalityzation: UI Personalization, Theoretical Grounding in HCI and Design Research. *AIS Transactions on Human-Computer Interaction* 7, 2 (2015), 43–69. https://aisel.aisnet.org/thci/vol7/iss2/1

[4] Mary Beth Kery and Brad A. Myers. 2017. Exploring exploratory programming. In *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 25–29. https://doi.org/10.1109/VLHCC.2017.8103446

[5] Yuzhe Cai, Shaoguang Mao, Wenshan Wu, Zehua Wang, Yaobo Liang, Tao Ge, Chenfei Wu, Wang You, Ting Song, Yan Xia, Jonathan Tien, Nan Duan, and Furu Wei. 2024. Low-code LLM: Graphical User Interface over Large Language Models. arXiv:2304.08103 [cs.CL] https://arxiv.org/abs/2304.08103

[6] Jonathan Chen and Dongwook Yoon. 2024. Exploring the Diminishing Allure of Paper and Low-Fidelity Prototyping Among Designers in the Software Industry: Impacts of Hybrid Work, Digital Tools, and Corporate Culture. In *Proceedings of the CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '24)*. Association for Computing Machinery, New York, NY, USA, Article 1024, 14 pages. https://doi.org/10.1145/3613904.3642774

[7] Nigel Cross. 2004. Expertise in design: an overview. *Design Studies* 25, 5 (2004), 427–441. https://doi.org/10.1016/j.destud.2004.06.002 Expertise in Design.

[8] Luiz Fernando C.S. Durão, Kevin Kelly, Davi N. Nakano, Eduardo Zancul, and Conor L. McGinn. 2018. Divergent prototyping effect on the final design solution: the role of "Dark Horse" prototype in innovation projects. *Procedia CIRP* 70 (2018), 265–271. https://doi.org/10.1016/j.procir.2018.03.278 28th CIRP Design Conference 2018, 23-25 May 2018, Nantes, France.

[9] Steven P. Dow, Alana Glassco, Jonathan Kass, Melissa Schwarz, Daniel L. Schwartz, and Scott R. Klemmer. 2011. Parallel prototyping leads to better design results, more divergence, and increased self-efficacy. *ACM Trans. Comput.-Hum. Interact.* 17, 4, Article 18 (dec 2011), 24 pages. https://doi.org/10.1145/1879831.1879836

[10] Figma, Inc. 2023. Figma: Collaborative Interface Design Tool. https://www.figma.com. Accessed: 2024-09-12.

[11] GitHub. 2024. *GitHub Copilot.* https://github.com/features/copilot Accessed: 2024-09-11.

[12] Frederic Gmeiner and Nur Yildirim. 2023. Dimensions for Designing LLM-based Writing Support. In *In2Writing Workshop at CHI 2023.* https://aisel.aisnet.org/jais/vol12/iss11/2

[13] Google. 2023. *Google Gemini Tool.* https://www.google.com/gemini Accessed: 2024-09-11.

[14] Sandra G Hart and Lowell E Staveland. 1988. *Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research.* Technical Report NASA-TM-873301. NASA Ames Research Center, Moffett Field, CA. Available: https://humansystems.arc.nasa.gov/groups/tlx/.

[15] Irving L. Janis. 1982. *Groupthink: Psychological Studies of Policy Decisions and Fiascoes* (2nd ed.). Houghton Mifflin, Boston.

[16] David G. Jansson and Steven M. Smith. 1991. Design fixation. *Design Studies* 12, 1 (1991), 3–11. https://doi.org/10.1016/0142-694X(91)90003-F

[17] Ellen Jiang, Kristen Olson, Edwin Toh, Alejandra Molina, Aaron Donsbach, Michael Terry, and Carrie J Cai. 2022. PromptMaker: Prompt-based Prototyping with Large Language Models. In *Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems* (New Orleans, LA, USA) *(CHI EA '22)*. Association for Computing Machinery, New York, NY, USA, Article 35, 8 pages. https://doi.org/10.1145/3491101.3503564

[18] Bo Kang, Nathan Crilly, Weining Ning, and Per Ola Kristensson. 2023. Prototyping to elicit user requirements for product development: Using head-mounted augmented reality when designing interactive devices. *Design Studies* 84 (2023), 101147. https://doi.org/10.1016/j.destud.2022.101147

[19] Mary Beth Kery, Amber Horvath, and Brad A. Myers. 2017. Variolite: Supporting Exploratory Programming by Data Scientists. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. ACM, 1265–1276. https://doi.org/10.1145/3025453.3025626

[20] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2023. Large Language Models are Zero-Shot Reasoners. arXiv:2205.11916 [cs.CL]

[21] Yujia Li, David Choi, Junyoung Chung, Jeremiah Z Liu, David Krueger, Sasha Rush, Oriol Vinyals, et al. 2022. Competition-Level Code Generation with AlphaCode. https://www.deepmind.com/publications/competitive-programming-with-alphacode. Accessed: 2024-09-12.

[22] David Chuan-En Lin and Nikolas Martelaro. 2024. Jigsaw: Supporting Designers to Prototype Multimodal Applications by Chaining AI Foundation Models. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*. 4:1–4:15. https://doi.org/10.1145/3613904.3641920

[23] Yuwen Lu, Ziang Tong, Qinyi Zhao, Chengzhi Zhang, and Toby Jia-Jun Li. 2023. UI Layout Generation with LLMs Guided by UI Grammar. arXiv:2310.15455 [cs.HC] https://arxiv.org/abs/2310.15455

[24] Stephen MacNeil, Johanna Okerlund, and Celine Latulipe. 2017. Dimensional Reasoning and Research Design Spaces. In *Proceedings of the 2017 ACM SIGCHI Conference on Creativity and Cognition* (Singapore, Singapore) *(C&C '17)*. Association for Computing Machinery, New York, NY, USA, 367–379. https://doi.org/10.1145/3059454.3059472

[25] Ference Marton and Shirley Booth. 1997. *Learning and Awareness* (1st ed.). Routledge, New York. 240 pages. https://doi.org/10.4324/9780203053690 arXiv:9780203053690

[26] Meta AI. 2023. *Code Llama: An AI Tool for Coding.* https://ai.meta.com/blog/code-llama-large-language-model-coding/ Accessed: 2024-09-11.

[27] Meta Platforms, Inc. 2013. React: A JavaScript library for building user interfaces. https://react.dev. Version 18.0, accessed September 11, 2024.

[28] MUI Core Contributors. 2014. Material-UI: React components for faster and easier web development. https://mui.com. Version 5.0, accessed September 11, 2024.

[29] Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lianmin Zheng, Zi Lin, Shafiq Joty, Hao Ma, and Caiming Xiong. 2022. CodeGen: An Open Large Language Model for Code with Multi-Turn Program Synthesis. https://github.com/salesforce/CodeGen. Accessed: 2024-09-12.

[30] OpenAI. 2023. GPT-4: OpenAI's Large Language Model. https://openai.com/research/gpt-4. Accessed: 2024-09-12.

[31] Savvas Petridis, Nicholas Diakopoulos, Kevin Crowston, Mark Hansen, Keren Henderson, Stan Jastrzebski, Jeffrey V Nickerson, and Lydia B Chilton. 2023. AngleKindling: Supporting Journalistic Angle Ideation with Large Language Models. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems* (Hamburg, Germany) *(CHI '23)*. Association for Computing Machinery, New York, NY, USA, Article 225, 16 pages. https://doi.org/10.1145/3544548.3580907

[32] Pythagora.io. 2024. *GPTPilot.* https://github.com/Pythagora-io/gpt-pilot Accessed: 2024-09-11.

[33] Yijun Qian, Yujie Lu, Alexander Hauptmann, and Oriana Riva. 2024. Visual Grounding for User Interfaces. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 6: Industry Track)*, Yi Yang, Aida Davani, Avi Sil, and Anoop Kumar (Eds.). Association for Computational Linguistics, Mexico City, Mexico, 97–107. https://doi.org/10.18653/v1/2024.naacl-industry.9

[34] Ben Shneiderman. 2007. Creativity support tools: accelerating discovery and innovation. *Commun. ACM* 50, 12 (dec 2007), 20–32. https://doi.org/10.1145/1323688.1323689

[35] Chenglei Si, Yanzhe Zhang, Zhengyuan Yang, Ruibo Liu, and Diyi Yang. 2024. Design2Code: How Far Are We From Automating Front-End Engineering? arXiv:2403.03163 [cs.CL] https://arxiv.org/abs/2403.03163

[36] Davit Soselia, Khalid Saifullah, and Tianyi Zhou. 2023. Learning UI-to-Code Reverse Generator Using Visual Critic Without Rendering. arXiv:2305.14637 [cs.CV] https://arxiv.org/abs/2305.14637

[37] Sangho Suh, Meng Chen, Bryan Min, Toby Jia-Jun Li, and Haijun Xia. 2024. Luminate: Structured Generation and Exploration of Design Space with Large Language Models for Human-AI Co-Creation. In *Proceedings of the CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '24)*. Association for Computing Machinery, New York, NY, USA, Article 644, 26 pages. https://doi.org/10.1145/3613904.3642400

[38] Mirac Suzgun, Nathan Scales, and Nathanael Schärli. 2022. Challenging BIG-Bench Tasks and Whether Chain-of-Thought Can Solve Them. arXiv:2210.09261 [cs.CL]

[39] Edward Tiong, Olivia Seow, Bradley Camburn, Kenneth Teo, Arlindo Silva, Kristin Wood, Daniel D. Jensen, and Maria C. Yang. 2019. The Economies and Dimensionality of Design Prototyping: Value, Time, Cost, and Fidelity. *Journal of Mechanical Design* 141, 3 (2019), 031105. https://doi.org/10.1115/1.4042337

[40] Joseph Tranquillo. 2015. Coding To Think: Teaching Algorithmic Thinking from Idea to Code. *Journal of Engineering Education Transformations* 28, 4 (2015), 23–32. https://digitalcommons.bucknell.edu/fac_journ/1312/

[41] Priyan Vaithilingam, Elena L. Glassman, Jeevana Priya Inala, and Chenglong Wang. 2024. DynaVis: Dynamically Synthesized UI Widgets for Visualization Editing. arXiv:2401.10880 [cs.HC] https://arxiv.org/abs/2401.10880

[42] Yuxuan Wan, Chaozheng Wang, Yi Dong, Wenxuan Wang, Shuqing Li, Yintong Huo, and Michael R. Lyu. 2024. Automatically Generating UI Code from Screenshot: A Divide-and-Conquer-Based Approach. arXiv:2406.16386 [cs.SE] https://arxiv.org/abs/2406.16386

[43] Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. 2024. Opendevin: An open platform for ai software developers as generalist agents. *arXiv preprint arXiv:2407.16741* (2024).

[44] WebSim.ai. 2023. WebSim: AI-powered Simulation Platform. https://websim.ai. Accessed: 2024-09-12.

[45] Jason Wei, Xuezhi Wang, and Dale Schuurmans. 2022. Chain of Thought Prompting Elicits Reasoning in Large Language Models. *CoRR* abs/2201.11903 (2022). arXiv:2201.11903 https://arxiv.org/abs/2201.11903

[46] Stephan Wensveen and Ben Matthews. 2014. Prototypes and Prototyping in Design Research. In *The Routledge Companion to Design Research* (1st ed.). Routledge, 15. arXiv:9781315758466

[47] Jason Wu, Eldon Schoop, Alan Leung, Titus Barik, Jeffrey P. Bigham, and Jeffrey Nichols. 2024. UICoder: Finetuning Large Language Models to Generate User Interface Code through Automated Feedback. arXiv:2406.07739 [cs.CL] https://arxiv.org/abs/2406.07739

[48] Tongshuang Wu, Ellen Jiang, Aaron Donsbach, Jeff Gray, Alejandra Molina, Michael Terry, and Carrie J Cai. 2022. PromptChainer: Chaining Large Language Model Prompts through Visual Programming. arXiv:2203.06566 [cs.HC] https://arxiv.org/abs/2203.06566

[49] Tongshuang Wu, Michael Terry, and Carrie J. Cai. 2021. AI Chains: Transparent and Controllable Human-AI Interaction by Chaining Large Language Model Prompts. *CoRR* abs/2110.01691 (2021). arXiv:2110.01691 https://arxiv.org/abs/2110.01691

[50] Shuhong Xiao, Yunnong Chen, Jiazhi Li, Liuqing Chen, Lingyun Sun, and Tingting Zhou. 2024. Prototype2Code: End-to-end Front-end Code Generation from UI Design Prototypes. arXiv:2405.04975 [cs.SE] https://arxiv.org/abs/2405.04975

[51] Ann Yuan, Andy Coenen, Emily Reif, and Daphne Ippolito. 2022. Wordcraft: Story Writing With Large Language Models. In *Proceedings of the 27th International Conference on Intelligent User Interfaces* (Helsinki, Finland) *(IUI '22)*. Association for Computing Machinery, New York, NY, USA, 841–852. https://doi.org/10.1145/3490099.3511105

[52] Sukmin Yun, Haokun Lin, Rusiru Thushara, Mohammad Qazim Bhat, Yongxin Wang, Zutao Jiang, Mingkai Deng, Jinhong Wang, Tianhua Tao, Junbo Li, Haonan Li, Preslav Nakov, Timothy Baldwin, Zhengzhong Liu, Eric P. Xing, Xiaodan Liang, and Zhiqiang Shen. 2024. Web2Code: A Large-scale Webpage-to-Code Dataset and Evaluation Framework for Multimodal LLMs. arXiv:2406.20098 [cs.CV] https://arxiv.org/abs/2406.20098

[53] J.D. Zamfirescu-Pereira, Richmond Y. Wong, Bjoern Hartmann, and Qian Yang. 2023. 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* (Hamburg, Germany) *(CHI '23)*. Association for Computing Machinery, New York, NY, USA, Article 437, 21 pages. https://doi.org/10.1145/3544548.3581388

A Few-shot Examples for the Design Matrix

<Few-shot Examples for the Design Matrix>

OKCupid -
PersonXIdea: Single person
PersonXGrounding: - Difficulty in finding potential partners who meet specific criteria like race, religion, age, or occupation. \n - Challenges in efficiently filtering through dating apps to locate compatible matches. \n - Need for an application that streamlines the process by allowing users to set precise criteria and receive curated suggestions.
ApproachXIdea: Searchable Database to allow people to search for people based on specified criteria
ApproachXGrounding: - Ensure the database includes comprehensive filters such as age, gender, sex, religion, and occupation to meet users' specific search criteria. \n - Develop a robust and scalable search algorithm that efficiently handles large datasets and returns accurate results based on the selected filters. \n - Implement user-friendly search and filtering interfaces that make it easy for users to apply multiple criteria and refine their search results.
InteractionXIdea: Faceted Browsing
InteractionXGrounding: - Provide users with multiple facet filters, including age, gender, religion, occupation, and location, to refine their search effectively. \n - Ensure the UI dynamically updates search results in real-time as users adjust their facet filters, offering immediate feedback. \n - Design intuitive navigation with clear options to reset filters, switch criteria, and save searches, using checkboxes, sliders, and dropdowns for ease of use.

Tinder:
PersonXIdea: Single person
PersonXGrounding: - Users often struggle to find matches that meet their physical preferences quickly and efficiently. \n - The abundance of profiles makes it challenging to identify those that align with specific looks or appearances. \n - A streamlined approach to swiping and filtering by appearance would help users connect with potential matches faster, focusing on visual attraction.
ApproachXIdea: Lower the cognitive load by providing less information, making it easier to judge potential matches quickly
ApproachXGrounding: - Limit the displayed information to essential details, such as a single profile photo and a brief tagline, to encourage snap judgments. \n - Focus on visual appeal as the primary matching criterion, reducing the need for users to sift through extensive profiles. \n - Use an algorithm to prioritize matches based on visual preferences and minimal data inputs, streamlining the matching process.
InteractionXIdea: Card Swipe
InteractionXGrounding: - Each card should prominently feature a large profile photo, as visual appeal is the primary factor in this interaction. \n - Include minimal text, such as the person's name, age, and a short tagline or fun fact, to provide just enough context without overwhelming the user. \n - Add simple icons or buttons for actions like "Like" or "Pass," ensuring that users can quickly swipe or tap to make their choice.

Coffee Meets Bagel:
PersonXIdea: Single person
PersonXGrounding: - Users who are looking for serious relationships prefer fewer, high-quality matches over endless swiping. \n - The overwhelming number of potential matches on other apps can make it difficult to focus on finding a meaningful connection.\n - An app designed for serious dating should streamline the process by offering a curated selection of potential partners, reducing time spent on the app.
ApproachXIdea: Lower the cognitive load by having less matches to make more intentional judgements
ApproachXGrounding: - Present a select number of potential matches each day to prevent decision fatigue and promote thoughtful consideration.\n - Display key information like shared interests, compatibility indicators, and mutual friends to aid decision-making without overwhelming the user. \n - Prioritize quality over quantity, ensuring that each match is relevant to the user's preferences and relationship goals.
InteractionXIdea: Feed with 5 options to date
InteractionXGrounding: - The daily message should include a concise profile summary for each of the five matches, highlighting essential details such as name, age, occupation, and a short personal note or shared interest. \n - Include compatibility scores or commonalities (e.g., mutual friends, hobbies) to help users quickly assess each match's potential. \n - Provide clear action buttons within the message to either like, pass, or start a conversation, making it easy for users to engage with their daily options.

Figure 8: Few-shot examples for entries in the Design Matrix

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009