Evaluation of Code Generation for Simulating Participant Behavior in Experience Sampling Method by Iterative In-Context Learning of a Large Language Model

Evaluation of Code Generation for Simulating Participant Behavior in Experience Sampling Method by Iterative In-Context Learning of a Large Language Model

ALIREZA KHANSHAN, Industrial Design Department of Eindhoven University of Technology, Netherlands
PIETER VAN GORP, Industrial Engineering Department of Eindhoven University of Technology, Netherlands
PANOS MARKOPOULOS, Industrial Design Department of Eindhoven University of Technology, Netherlands

The Experience Sampling Method (ESM) is commonly used to understand behaviors, thoughts, and feelings in the wild by collecting self-reports. Sustaining sufficient response rates, especially in long-running studies remains challenging. To avoid low response rates and dropouts, experimenters rely on their experience, proposed methodologies from earlier studies, trial and error, or the scarcely available participant behavior data from previous ESM protocols. This approach often fails in finding the acceptable study parameters, resulting in redesigning the protocol and repeating the experiment. Research has shown the potential of machine learning to personalize ESM protocols such that ESM prompts are delivered at opportune moments, leading to higher response rates. The corresponding training process is hindered due to the scarcity of open data in the ESM domain, causing a cold start, which could be mitigated by simulating participant behavior. Such simulations provide training data and insights for the experimenters to update their study design choices. Creating this simulation requires behavioral science, psychology, and programming expertise. Large language models (LLMs) have emerged as facilitators for information inquiry and programming, albeit random and occasionally unreliable. We aspire to assess the readiness of LLMs in an ESM use case. We conducted research using GPT-3.5 turbo-16k to tackle an ESM simulation problem. We explored several prompt design alternatives to generate ESM simulation programs, evaluated the output code in terms of semantics and syntax, and interviewed ESM practitioners. We found that engineering LLM-enabled ESM simulations have the potential to facilitate data generation, but they perpetuate trust and reliability challenges.

CCS Concepts: • **Computing methodologies** → **Simulation types and techniques**; • **Software and its engineering** → *Collaboration in software development*; • **Human-centered computing** → **Interactive systems and tools**.

Additional Key Words and Phrases: Experience Sampling Method, Large Language Model, Behavior Simulation, Prompt Engineering

ACM Reference Format:
Alireza Khanshan, Pieter Van Gorp, and Panos Markopoulos. 2024. Evaluation of Code Generation for Simulating Participant Behavior in Experience Sampling Method by Iterative In-Context Learning of a Large Language Model. *Proc. ACM Hum.-Comput. Interact.* 8, EICS, Article 255 (June 2024), 19 pages. https://doi.org/10.1145/3661143

Authors' Contact Information: Alireza Khanshan, Industrial Design Department of Eindhoven University of Technology, Eindhoven, Netherlands, a.khanshan@tue.nl; Pieter Van Gorp, Industrial Engineering Department of Eindhoven University of Technology, Eindhoven, Netherlands, p.m.e.v.gorp@tue.nl; Panos Markopoulos, Industrial Design Department of Eindhoven University of Technology, Eindhoven, Netherlands, p.markopoulos@tue.nl.

ACM 2573-0142/2024/6-ART255
https://doi.org/10.1145/3661143

1 INTRODUCTION

The Experience Sampling Method (ESM) is widely used in a variety of domains such as psychology [58], healthcare [36], and human-computer interaction (HCI) [13, 27] to understand people's thoughts, behaviors, and feelings through self-reporting [15, 25, 36]. ESM provides high reliability and validity by spreading notifications throughout the study days, aiming to provide an ecologically valid data collected in real-life conditions concerning the research question(s) [15, 25, 36]. Low response rates and dropouts especially in ESM studies that last more than a few weeks are well-documented challenges in the domain [46, 57]. Typical causes relate to receiving notifications at inopportune moments [23] and a high number of notifications per day [17, 61] which can result in frustrations and response fatigue for study participants [46, 57].

ESM experimenters could potentially benefit from leveraging historical participant behavior data to learn from the impact of study design choices on participation and in turn, could refine their research protocols. However, the domain lacks such open-access protocol-participation data [33]. Machine learning methods in combination with the ample computational capacity of in-situ ESM devices (e.g., smartphones [60] and smartwatches [32]) could learn the opportune moments of notification delivery and optimize the number of prompts on the fly and over time [39, 45, 62, 63]. At the moment, such learning process can only start without any prior knowledge, due to data deficiency in the domain, which would fall back to randomly-timed notifications at the beginning of the learning process; a well-know problem called cold start [59]. A cold start can be partially addressed through a simulation that can generate sufficient data for pre-training [33, 66] (see Fig. 1). Such a simulation can also provide valuable insights for experimenters to approximate the expected outcome behaviors to adjust their protocol. The relevance of such user simulations is further demonstrated especially in HCI research [24, 33, 50]. In the ESM context, we specifically focus on human behavior simulators that imitate human responses to notifications [33]. Utilizing such simulators requires domain knowledge related to human behavior but also requires programming skills whenever adaptations, beyond what the tool offers, are needed. To alleviate such burden and make synthetic ESM data generation more accessible, we aim to explore the capabilities of the GPT-3.5 turbo-16k large language model (LLM) to assess the feasibility of applying a state-of-the-art LLM for creating behavior simulations that could aid experimenters with their ESM research.

LLMs are currently revolutionizing software development practices by enabling developers to shift from scavenging the internet for code extracts to instructing LLMs to generate and iteratively refine code according to their needs. While this approach holds the potential for increased efficiency, it also poses new methodological challenges for software engineering research. This development could serve as a catalyst for realizing the longstanding vision of end-user development [5, 51], allowing individuals without formal software expertise to independently create customized code. Our exploration focuses on the feasibility of this approach in the ESM domain where advanced engineering expertise is often required, but the end-users themselves (such as social scientists and psychologists) possess domain-specific knowledge without necessarily being familiar with software engineering principles.

We explore in-context learning [42] to generate ESM-specific simulation programs that would otherwise demand a deep understanding of human behavior and programming. We explore various prompt design alternatives to assess the quality of generated code in terms of semantics and syntax. To facilitate this process we created a helper tool called "Prompt4Code" to generate domain-specific code by automatically augmenting the input prompts (from a non-technical domain expert) with several technically detailed prompts extracted from code examples (made by technical experts) [3]. We then demonstrated our findings to domain experts during several interview sessions, let them interact with Prompt4Code, and discussed the outcomes. Although other LLMs such as Google's

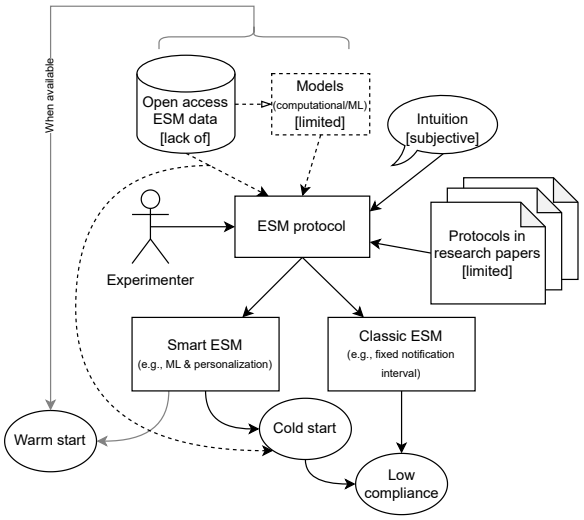

Fig. 1. The current resources that experimenters typically rely on in the ESM domain often result in sub-optimal protocols. While open access ESM data and derived machine learning models could help fine-tune such protocols to increase engagement.

PaLM 2 or Meta's LLaMA 2 could also be used for our research, we opted for OpenAI's GPT-3.5 for its ease of access, open API, and popularity. The performance comparison between different LLMs lies outside the scope of this paper.

In the following sections, we overview the background leading to the current paper and relevant contributions in this domain. Then, we introduce Prompt4Code, our applied methods to create prompts, and our step-wise approach from zero-shot to few-shot learning. Subsequently, in the results sections we elaborate on our experimental setup, inputs, and outputs in each step, then report on our conducted structured interviews with ESM experts and practitioners. In the discussion section, we debate the implications of our findings and reassess the limitations of LLMs as well as our approach. Lastly, we conclude by summarizing our findings and invite the readers to investigate potential directions for future work.

2 BACKGROUND

We review the principles leading up to the developments presented in this paper. Since we aspire to make user behavior simulation more accessible to ESM practitioners, we overview the research on utilizing LLMs for programming in general but also in HCI research and interactive engineering.

LLMs help us with a variety of tasks from language translation [67] to writing code [22]. An LLM is a language model transformer with hundreds of billions or even more parameters, which are trained on massive text data [55] such as GPT-3 [9], Palm [12], and more recently GPT-4 [48]. LLMs are scaled successors of pre-trained language models (PLMs) [55]. Through such scaling, GPT-3, for example, can already solve few-shot tasks through in-context learning, whereas GPT-2, its predecessor, failed [67]. In natural language processing, in-context learning, few-shot learning, or few-shot prompting is a technique for conversing with LLM in natural language to get a response to an inquiry (prompt). In-context learning allows a model to process examples before attempting to generate a response to a prompt [42]. The performance of LLMs depends on the examples provided for in-context learning [54].

Popularized use of LLMs such as in ChatGPT [1] raises high expectations concerning their capabilities in task automation and writing code that could potentially offload the burden of software development. End users can generate code by conversing in natural language. However, it is important to have sufficient knowledge, experience, and expertise to converse effectively such that the generated response is acceptable. Automatically augmenting natural language input with appropriate content such that the ultimate prompt increases the likelihood of producing high-quality code sounds plausible. Such augmentation enables end users to converse with an LLM needless of prompt engineering knowledge. Prompt engineering refers to applying a variety of techniques to converse effectively with LLMs [64]. Although the potential of prompt engineering to generate software applications is shown (e.g., [56]), there is insufficient scientific knowledge of engineering such prompts for building LLM-based code generators as facilitators for non-technical domain experts. We have only begun to explore the potential implications of integrating such an emergent technology in well-established software engineering processes.

Although LLMs are the backbone of the emergent generation of AI assistants, they are known to *hallucinate* [1]: the generated output may sound plausible and realistic but factually incorrect [4]. Therefore, LLM-generated response, be it source code or not, cannot be blindly trusted. To mitigate such hallucinations, incorporating external knowledge and feedback has been effective [52]. Even though LLM-based code assistants are perceived to be aiding with productivity [11, 19, 53], the generated code often falls below standards of secure coding [18, 34], calling for further exploration and investigations both for prompt-response evaluation and inspecting the generated code in solving complex tasks [41]. Nevertheless, LLMs are being researched and utilized in professional settings as programming assistants [2, 16, 22, 53, 65]. Similarly, leveraging LLMs is gaining popularity in HCI research and engineering interactive systems. For example in simulating social behaviors [50], generating synthetic questionnaire and survey responses [24, 28], and customizing and styling the user interface for web pages with natural language [35, 38].

Although we treat LLMs in such applications as black boxes, we have full control over how we prompt. Prompt engineering is emerging as the skill set needed to converse effectively with LLMs [64]. Exploring prompt design alternatives for solving problems in different domains could help define guidelines to enhance the output of LLMs [64]. By applying prompting strategies such as providing examples, or asking the LLM to adopt a persona [49] we can improve its output to match our desired outcome through iteration and elaboration. Strategies for guiding specific programming tasks [35, 38, 56], applying LLM to scientific surveys [24], or simulating social behaviors [50] have been attempted before. Such LLM-enabled explorations in HCI research show the potential of such applications in ESM, where domain experts can potentially benefit from utilizing LLMs to refine their ESM protocols, although it has not been applied before.

3 METHODS

Scheduling notifications and understanding how users interact with them are crucial aspects of ESM research. Such notifications are typically delivered on smartphones [60]. When receiving a notification on a smartphone, if the notification is noticed, the user will either engage with it or neglect it depending on a variety of factors, e.g., the task at hand or the environment the user is in. Simulating such behavior could be particularly helpful in optimizing the notification schedule for maximizing user engagement, e.g., to ensure a higher response rate in a survey by allowing researchers to approximate how burdensome their research protocol could be, and estimate dropout and response rates. As a result, researchers can adjust their protocol, test it in simulation, and then run it in the wild, hoping for a higher response rate and fewer dropouts [33].

We programmed our simulator integrated with a process-based discrete-event simulation framework (Simpy [21, 44]) which is a common and popular approach in human behavior simulation [43].

In our simulation, we modeled computational psychological theories about memory accessibility and motivation and incorporated personal context; a combination that can capture a variety of use cases and synthesize human behavior similar to the observed behaviors during real-life ESM experiments [31, 33]. The goal of this simulator was to be used by ESM experimenters to simulate the response behavior (i.e., opening or ignoring notifications) of their cohort given the research protocol (e.g., the number of notifications per day). After several interactions with domain experts in HCI, social, and behavioral sciences (potential end-user experimenters) we were encouraged to further develop the simulator so it is generic enough to allow the integration of a variety of external or user-tailored theories, models, and historical data. But also more accessible such that less or no coding would be required. Given the recent technological achievements, we were intrigued to explore LLM-enabled variations of the simulator. Such an approach for data generation, if successful, could be used by experimenters to fine-tune their study design parameters and could be used for training machine learning models to avoid the cold start problem as elaborated in Section 1.

In the following subsections, we explain the methods applied, including our approach for efficient LLM-based code generation using our custom-made tool, Prompt4Code, prompt engineering, and interviews with experts experienced in ESM research or similar methods.

3.1 Prompt4Code: A Helper Tool for In-Context Learning and Code Generation

To make our simulator more accessible such that experimenters could create ESM simulation programs with natural language rather than programming, we aim to train GPT-3.5 turbo-16k by feeding our hand-crafted ESM simulation codes as examples. In turn, we expect to output similarly structured simulation programs. To prompt an LLM such that it can generate output based on some prior knowledge (in our case ESM simulation), a set of prompts as examples should be fed to the model to be processed before the ultimate output is made.

To optimize this process, also known as in-context learning (see Section 2), we created Prompt4Code, a command line tool to automatically augment a user prompt with technically informative examples tailored for code generation. The tool traverses a directory of choice including source code files as examples, then extracts docstrings and comments from source files and turns them into instructive prompts using pre-defined templates in natural language.

Prompt4Code was created with the idea of allowing fast prototyping of ESM simulation programs when the ESM expert does not want to get involved in rigorous simulation programming while the output code can be fixed or modified by programmers who are not necessarily experts in ESM. This could be ideal in small teams that involve non-programmer ESM experts and programming assistants such as junior programmers or student assistants (see Fig. 2).

To initiate a conversation and generate prompts, the user must provide one or more sentences describing their ESM simulation. This prompt serves as the starting point for generating source code but is augmented in the background with auto-generated prompts derived from the source code examples directory. The user has the flexibility to choose a specific LLM model to be used for prompt generation by providing the model name or identifier. This allows the user to tailor the code generation process based on their requirements and preferences. As mentioned earlier we used `gpt-3.5-turbo-16k` in our specific experiments. Additionally, Prompt4Code can limit the number of input files used for in-context learning. By specifying a limit, the user can control the size of the dataset used for learning, which can be beneficial for performance optimization and specifically useful considering token limitation and saving costs when using paid or commercial models. To explore the learning variations, especially when an input limit is set, users can choose to shuffle the order of the list of example files. Prompt4Code allows users to immediately run and display the generated code. Executing the generated code and a verbose output enables the user to quickly assess the functionality and correctness of the generated code. The latest documentation

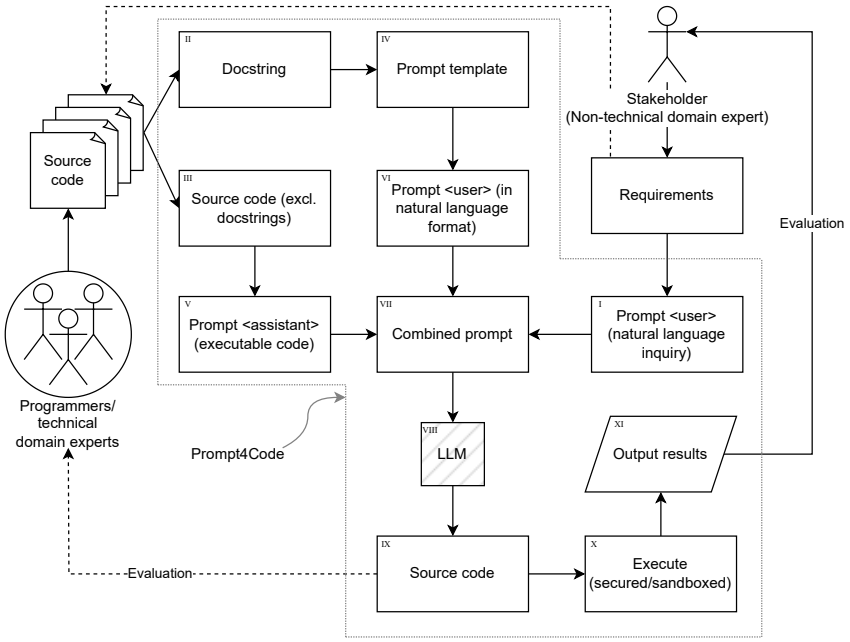

Fig. 2. Prompt4Code dissects a directory of choice, separates docstrings and comments from source code, and orchestrates in-context learning based on predefined natural language templates. A [non-technical] domain expert can express their desired program in natural language. To increase the quality of the output the expert provides a source code directory. Such codes are usually crafted by technical experts in the same domain. The order of task execution within Prompt4Code is represented by Roman numbers in the figure.

and detailed user manual of Prompt4Code with a full list of its commands is available online at https://github.com/khnshn/prompt4code.

3.2 Step-wise approach

A beneficial approach to simulating human behavior is to imagine the personas and daily activities of the target population. By estimating the expected daily behaviors, an ESM notification schedule can be fine-tuned to send notifications when events of interest occur or to avoid sending notifications when other higher-priority tasks are at hand. To drive our simulations, we require a scenario that involves human subjects, represents real-life conditions, and the day-to-day activities can be modeled as discrete events. We selected a simplified yet recurring case study from the ESM literature which studies school teachers [6, 7, 29, 30] and with limited states (e.g., teaching, and on a break) in a prompt that expresses their daily routines and moments of interest:

"Simulate teachers' activities during a day. Teachers will receive notifications on their smartphones every 2 hours. Then depending on their situation they either respond to the notification or ignore it. The program should estimate the response rate to such notifications for 50 teachers."

We focus on assessing the capability of LLM-based code generation to estimate the compliance of participants to an ESM protocol in terms of response rate. The example prompt above and the resulting generated code lays a baseline for subsequent prompt iterations in this paper. We comply with the best practices proposed by OpenAI to compose our prompts. We include details in our query to get more relevant answers, ask the model to 'adopt a persona', and 'provide examples' to

perform more accurate calculations [49]. To ask the model to adopt a persona, a *system* message should be composed. We also explored scenarios where such system prompts were excluded but they led to inferior-quality output ergo not reported in this paper. A system message is incorporated in Prompt4Code by default [3]:

"You are a helpful assistant that generates Python code per request. Your output only includes executable source code. Your output does not include any sort of explanation or description."

We also followed the 'code execution practice' proposed by OpenAPI. We executed the very response of the model. As previously mentioned, models may hallucinate and previous research has also confirmed the risks of running generated code as-is since it might have security vulnerabilities. To mitigate that, the code execution needs to be sandboxed. For simplicity and flexibility, instead of sandboxing, we limited the built-in modules that can be accessed via Prompt4Code although the user can add or remove modules based on their requirements.

To explore the capabilities of GPT-3.5 turbo-16k, first, we did a set of zero-shot prompts, which means no example was provided to generate the code. Then, we did one-shot prompts by providing one code sample. This necessitated the use of *assistant* message as demanded by the OpenAI. An assistant message contains the expected output (the code sample) to the prompt (*user* message). Then we explored the same approach but instead of prompting for a whole program, we requested a code snippet and a configuration dictionary. We did so to investigate the performance of the LLM when more constraints are present and the request is less complex. Such constraints should discourage the model from generating a full program but rather a snippet such as the body of a function with certain specifications or a structured dictionary that configures a simulation scenario (see Section 4). Lastly, we utilized Prompt4Code to automatically augment code samples with accompanying prompts to facilitate more comprehensive in-context learning. In each step, we repeated the process to observe the degree to which the output of the model would vary. We stopped iterating over the same strategy as soon as a similar output was generated in a successive iteration.

3.3 Interview Method

To gather qualitative data on user experiences, perspectives, and feedback on the ESM simulation and the use of LLMs, we recruited $N = 4$ participants through convenience sampling, who were invited to participate in structured interviews aimed at understanding their perspectives on ESM and the challenges associated with it. Recruitment involved reaching out to researchers in the network with affiliations to ESM and similar methods. Despite the need for a larger sample size to reach saturation, time constraints precluded further recruitment or additional interviews. However, the recruited participants showcased diversity in their backgrounds, roles, and experiences which mostly covered the participant characteristics required to conduct the interviews. All of our participants had programming experience limited to statistical analysis with R and were not experienced in tool building and software engineering. The participants were a Ph.D. student with a background in biomedical science and human physiology, a Ph.D. researcher in health psychology and behavioral science, an assistant professor in medical biology, and an associate professor in Human-Technology Interaction (HTI), psychology, and well-being.

The structured interview comprised three parts. The first part introduced ESM and its challenges, discussing how simulation and synthetic data generation can address these challenges. Questions in this part focused on the participants' background, current research activities, experience with ESM, their wishes and challenges in ESM research, and their coding experience. The second part introduced the ESM simulation (without an LLM), inviting participants for feedback and insights. The third part involved an introduction to the few-shot LLM-enabled simulation tool, Prompt4Code,

and its performance (Section 4.4). Participants explored example prompts, created their own, and provided feedback on their preferences for interaction and potential applications.

Interviews took place either online or in person, in a meeting room on the campus of the Eindhoven University of Technology, using a laptop for demonstration purposes. The interviewer explained the three parts of the interview, ensuring clarity and consistency across sessions. Detailed notes were taken throughout the interviews, capturing participants' responses and insights. The source codes generated and prompts inputted during the sessions were run during the same session and we explored the outcome with the interviewees.

4 RESULTS

This section presents the inputs, outputs, and modifications made in each step during our prompt engineering process. The input and output of the examples below can be found online in https://github.com/khnshn/prompt4code/tree/main/responses. We provide a detailed overview of all the iterations and prompts during our assessment in the appendix in Table 1 and Table 2 respectively. We share our lessons learned from each prompting strategy, including zero-shot, one-shot, and few-shot, in subsequent subsections. Lastly, we share the results of our interviews with the domain experts.

4.1 Zero-shot

In the first iteration, we found that the code included inexecutable strings resulting in syntax errors. After manually removing such strings, the code was syntactically correct but did not calculate the response rate as requested in the prompt. The response behavior was purely random. Hence, it was semantically incorrect.

In a subsequent iteration, the code raised a value error due to the incorrect use of generators. Rather than bugfixing manually as in the previous step, we asked ChatGPT [47] (with GPT-3.5) to find the bug in the code and fix it but it claimed that the code did not have a bug. After conversing further with ChatGPT and directly pointing out the bug, it was able to fix the problem. However, the generated code did not calculate the response rate and also did not run for 50 teachers. Hence, it was semantically incorrect.

In another iteration the generated code resulted in unexpected behavior when looping through the teachers, skipped certain indices, simulated less than 50 teachers, and did not estimate the response rate. See rows 1-6 in Table 1 in the appendix for more details. After witnessing the inconsistencies in zero-shot iterations, we provided a simulation source code as an example to the LLM to further assess its capability as presented below.

4.2 One-shot (full code generation)

As a preliminary step towards generating potentially better-performing domain-specific code compared to the previous strategy, we provided a hand-crafted simulation code to enhance the understanding of LLM of our baseline prompt. The hand-crafted simulation was programmed to address the following user prompt:

"Write a Python program using the SimPy framework that simulates teachers' activities during a day. Teachers will receive notifications on their smartphones at random moments. Then, depending on their situation they either respond to the notification or ignore it. The program should simulate a single teacher in 1000 timesteps."

We paired the prompt above with an assistant prompt and the program that we handcrafted. After such modification, we observed that in its first iteration, the generated code had a syntax error since it used Enum as different states for teachers' behaviors while the Enum class was not defined. We asked ChatGPT to fix the bug and it was able to address the Enum-related issue. However, the

code had runtime errors due to syntactically wrong use of interruptions (a SimPy concept) and also was semantically incorrect.

In the following iteration, the Enum error persisted but ChatGPT was able to solve it this time. The program ran without any errors and the response rate was calculated as requested in the prompt. Since we provided an example utilizing a built-in SimPy interruptions mechanism, we expected to see a similar use of such a concept in the generated code. However, the output did not use such instructions even though they are a common technique used in such simulations.

In the last iteration of this type, the generated code still has a bug and attribute error due to incorrect object use. ChatGPT was able to fix the bug and output the response rate as requested. The code still did not use the interruptions mechanism which was exemplified in the one-shot example. See rows 7-10 in Table 1 in the appendix for more details.

4.3 One-shot (configuration generation)

Given the poor performance in the previous attempts, we explored asking for a configuration (e.g., a code snippet as in Listing 1) instead of a full executable program as detailed in rows 11-13 in Table 1 in the appendix. We prototyped an ESM response behavior simulator that can consume a configuration including declarations of what types of behaviors it should simulate (e.g., walking, teaching, eating), for how long, and whether such duration is fixed or random within a predefined interval. The configuration includes a *decision body*, which can be a flexible set of instructions that calculates, either stochastically or deterministically, whether an agent responds to a notification or not.

Listing 1. An example ESM configuration for simulating daily routines of study participants.

```
{
    "ticks": 1000,
    "agent": "teacher",
    "self-report": {"min": 2, "max": 5},
    "beep": {"interval_min": 50, "interval_max": 100},
    "decision_body": "self.i_want_to = True",
    "routines": [
        {
            "activity": "teaching",
            "busy": 90,
            "duration": {"type": "fixed", "value": 150},
        },
        {
            "activity": "breaking",
            "busy": 10,
            "duration": {"type": "random", "min": 5, "max": 10},
        },
    ],
}
```

Accordingly, we adjusted our user message to match the example above:

"Write a configuration dictionary in Python that specifies the execution of a simulation that runs for 1000 ticks, simulates a teacher. A teacher self-reports, and takes a random time between 2 to 5 ticks. teachers receive beeps randomly between every 50 to 100 ticks. The decision body refers to what a teacher considers before engaging (or not) with a beep. In this case, it should always be True. A teacher has two routines, teaching and taking a break. While teaching, a teacher is busy with a 90% chance, and the teaching duration is fixed for 150 ticks. A teacher is busy with a 10% chance when taking a break. The duration of a break is random between 5 to 10 ticks."

To assess whether the LLM could generate a configuration similar to the provided example with additional components, we created a similar user message with additional behaviors (i.e., walking to the break room and walking back to the classroom). If successful, we could potentially apply the same rationale to extend our configuration-based simulator with a conversational interface needless of hand-crafting the configuration and decision body. Therefore, we composed the user message as:

"Write a configuration dictionary in Python that specifies the execution of a simulation that runs for 4000 ticks, simulates a teacher. A teacher self-reports, and takes a random time between 1 to 4 ticks. Teachers receive beeps randomly between every 60 to 150 ticks. The decision body should be random, sometimes true, other times false. A teacher has four routines, teaching, walking to the break room, taking a break, and walking back to the classroom. The duration of each activity and how busy a teacher is during each activity are random."

The structure of the generated code fully complied with the provided example except for the decision body as it was not an executable Python code. Therefore, we adjusted our prompts by adding the following:

"...the decision body must be an executable Python code..."

The generated code remained syntactically correct, and the decision body turned executable, however, it did not follow the semantics implied by the given example. So, in our final iteration of this type, we updated our user prompt that precedes the assistant prompt by explicitly explaining the structure that the decision body should have. Such specific instruction is typically not expected to be given by a non-technical domain expert but could be given by a programmer:

"...The decision body, at the end of its procedure should set the value of self.i_want_to. In this case, it should always be True so self.i_want_to=True..."

With such modification, the output fully complied and the simulation that consumes such configurations ran flawlessly (see rows 11-13 of Table 1 in the appendix).

4.4 Few-shot

To facilitate a conversation with LLM including as many source code examples as possible while considering the token limitations we leveraged Prompt4Code (see Section 3.1). `gpt-3.5-turbo-16k` model with a 16,384 token limit (currently $0.003/1K for input tokens and $0.004/1K for output tokens) already allowed including all our example files which were in total 727 lines of code (including comments and docstrings) in 7 Python files covering different simulation scenarios. The examples included a set of docstrings explaining the purpose of each module, function, class, and argument along with the expected input and output of the program. Prompt4Code parses this information and separates the docstring from the source code to generate prompts as mentioned in section 3. By doing so each source code would have a matching prompt, explaining the purpose of the program. To save on the token usage, the comments and docstrings are removed from the source code and are instead present in the user prompt.

As detailed in the appendix, Table 1, in rows 14 and 15, the model output was more accurate compared to the other approaches in terms of semantics and syntax. Even in the first iteration, the generated code was clean, without any semantic or syntax error, ran as expected, and did output the response rate as requested in the prompt. Although promising, later iterations of the same prompt resulted in flawed code both semantically and syntactically. However, since the LLM was better tuned during in-context learning, a more comprehensive and structurally familiar simulation code was generated. Motivated by the aforementioned computational psychological theories that can enable human response behavior simulation (see Section 3 and [26]) we tried to prompt toward modeling motivation and attention, key factors in influencing a user's response:

"Write a Python program using SimPy that simulates teachers' activities during the day. Teachers will receive notifications on their smartphones every 2 hours. Then depending on their situation they either respond to the notification or ignore it. To better approximate the response behavior of teachers, take into consideration the attentional availability of the teachers and their level of motivation. The attentional availability should be calculated with the expected cost of interruption versus the benefit of giving a response. The level of motivation should be a linear function of time. For example, motivation levels might decrease at the end of the day. Accordingly, estimate the response rate to such notification. Run such a program for 50 teachers."

We witnessed that the model started to output less reliant results, containing both syntax and semantic errors. We specifically asked for more complex methods to calculate the human response behavior. i.e., by incorporating attentional availability and motivation. Although the code for calculating expected cost and linear increment were written, they were too simplistic. This indicates that on the one hand, the model might require more similar and explicit examples, and broken-down instructions to be able to *reason* about such concepts. On the other hand, a non-technical domain expert cannot rely on these outputs nor can always easily fix them hence we suggest the use of Prompt4Code in teams that also involve programmers. To further explore such impediments, we conducted interviews with ESM practitioners and domain experts in relevant fields, discussed our findings, and let them explore our simulation toolbox including the scenario with (Prompt4Code) and without LLM-enabled code generation.

4.5 Interviews

Through a thematic analysis after the interviews with $N = 4$ domain experts, we identified the following themes:

Trust in LLM-Based Code Generators Participants expressed concerns regarding the trustworthiness of LLMs in generating data. Specifically, the basis on which LLM generates data was questioned.

"I cannot [trust] the output of the LLM. It needs to be [explainable]." – P4

"Where does the [data] of the model come from?" – P1

Reliability and Consistency Reliability emerged as a significant theme, with participants discussing the need for consistent and reliable output from simulations. The level of output consistency varied, particularly when using LLM-based simulations compared to simulations without LLM.

"The output of the LLM is cool but I need to [validate] it with an [expert]" – P3

Privacy Concerns Privacy considerations, especially in the context of sharing ESM data, were voiced by participants. Privacy has become a critical factor, particularly when dealing with sensitive medical data.

"[Ethical approval] is strict in allowing data to be [open access]" – P1

Participants commonly shared apprehensions about trusting the output of LLM-based code generators. Despite skepticism, participants recognized the potential of simulators for various purposes, such as power analysis, user journey generation, and fine-tuning notification timing.

"For me, the simulation is less about outputting response rate but rather [monitoring] when the [motivation] to self-report drops" – P4

"Most behavioral scientists [don't code] at all, so the tool has interesting potential for them" – P2

All participants acknowledged challenges related to response rate and participation rate in ESM studies. The participants highlighted the benefits of using simulations for multiple purposes but not in all cases.

"I like to use the simulation to [find] the [optimal number] of notifications for my patients" – P3

"If I know all the details about what I want to simulate, I would [code it myself]" – P2

4.5.1 Quality of the Generated Code. During the interviews, the interviewer adopted the role of a technical expert with programming expertise and asked the participants for their desired simulation or data generation scenario. Participants P1 and P3 could come up with scenarios that could be formulated into respective prompts by the interviewer. The generated code for P1 did not have syntax errors and was executed successfully. However, the output was semantically flawed. The generated code for P3 had a syntax error and could not be fixed during the interview due to time constraints. Asking for simulation scenarios from P2 and P4 led to abstract yet critical conversations about the tool hence no prompt was formulated largely due to time constraints. All the participants acknowledged the potential of the approach and saw the benefit of using the tool. P4 especially appreciated prototyping in collaboration with programmers and students, a scenario that we imagined to be promising (see Section 3.1). The prompts and the outputs were stored for further analysis and future benchmarking and are available online at https://github.com/khnshn/prompt4code/tree/main/responses/interview.

5 DISCUSSION

LLMs exhibit instability as they undergo continuous development while offering the prospect of enhanced performance over time. Our research, along with prior work, highlights the need for a cautious interpretation of LLM output. Various concerns, including security, syntax, semantics, reusability, trust, consistency, and maintenance, warrant attention in the utilization of LLMs. Our investigation focused on evaluating the capabilities of GPT-3.5 turbo-16k to fast-track software prototyping for simulating ESM participant behavior. It is already well documented that GPT-3.5 turbo-16k and similar LLMs lack reasoning capabilities, yet we tried to let it reason in the most complex scenarios during our iterations. We consider it valuable to openly report these experiences, such that they can serve also as a benchmark example for future LLMs that will have improved reasoning capabilities, either built-in directly in the LLM or via external add-ons and plug-ins. The output of LLMs shows variability with each prompt, introducing inconsistencies. A prompt that once yielded secure, robust, and syntactically and semantically correct code could, after a retry, generate vulnerable and semantically flawed code, if not inexecutable. A meticulous review of the generated code, even when seemingly executing successfully, is essential to ensure semantic correctness and security. So it would be premature to assume that LLMs can enable end-user development at the moment. Literature suggests that end-user development is possible (e.g., [40] and [10]) but our study mainly focused on optimizing and automating code reuse which requires software engineering knowledge. Besides, the data underlying LLMs may not be up-to-date, which can pose challenges in practical implementation. We opted for GPT-3.5 turbo-16k, a state-of-the-art LLM that surpasses most benchmarks compared to other language models. Comparing other types of LLMs was regarded as a tangent since we focused on the feasibility testing of enabling non-programmer ESM practitioners to benefit from ESM simulation.

Despite these limitations, our research revealed that providing a greater number of code samples and employing systematic and automated prompt generation can optimize the efficiency of LLM utilization. Although the generated source code may possess inherent flaws, it could potentially serve as a foundation (boilerplate code or scaffolding) upon which developers can build and speed up the coding time. We interviewed the potential end-users (i.e., the ESM domain experts and experimenters) of our simulation. Despite voicing concerns regarding trust and reliability, it was acknowledged that the tool can help, especially junior programming assistants, for building ESM simulations that in turn could support study design choices such as determining sample size, number

of notifications per day, etc. Future work should assess such qualitative findings by quantifying the coding time and other relevant metrics to evaluate potential productivity gains.

It is important to note that Prompt4Code did not decompose its input prompt into smaller tasks. The docstring and comment parser in Prompt4Code utilized a static template and relied heavily on the content of the docstring and in-line comments. Thus, the quality of code documentation directly influences the quality of the augmented prompt. Leveraging natural language processing to ensure a coherent prompt, both semantically and grammatically could enhance the capability of such a tool. LLM-based code generators require fine-tuning and augmentation with external knowledge source(s) to produce more reliable output (e.g., via retrieval-augmented generation or RAG [20, 37]). Only then can we envision a reshaped future for development teams where (non-technical) domain experts can convert their requirements into a baseline prototype, through a natural language conversation, potentially quicker compared to current practice. In collaboration with technical experts in the same team, the prototype, as well as the code generator, can evolve simultaneously advancing more efficient software prototyping.

6 CONCLUSION

This research explored the potential of Large Language Models (LLMs) in automating code generation for supporting the Experience Sampling Method (ESM) protocols on smart mobile devices. By investigating prompt design alternatives in an ESM-specific case study, we evaluated the quality of the generated code to simulate human response behavior. Sustaining participation and achieving a higher response rate during ESM studies is a well-documented challenge [15, 61]. Synthesizing ESM response behavior data could help experimenters support or modify their study design choices to reduce participant dropouts. But also, can provide the necessary training data for enhancing ESM notification delivery that relies on machine learning for fine-tuning the timing and/or content of the notifications. We evaluated the capabilities of GPT-3.5 turbo-16k, a state-of-the-art LLM, in addressing our simulation problem. We compared LLM outputs in terms of semantics and syntax and discovered that our proposed automated in-context learning, which involves augmenting a user prompt with complementing technical prompts generated from source codes as examples, achieves higher quality code compared to other explored alternatives.

Our findings indicate that even though LLMs in general have the potential to assist developers in code writing, debugging, and refactoring; caution must be exercised in relying solely on their outputs, which aligns with previous research. Throughout our study, we observed that LLM outputs should be approached with skepticism due to various concerns. The outputs of LLMs are inconsistent, as the same prompt can result in contrasting code quality. Moreover, the generated code requires meticulous review for semantic correctness and security. Our research hints that providing more code samples and employing systematic and automatic prompt generation has the potential to save prototyping time for users of LLMs while enhancing the quality of the generated code.

Despite the flaws of the LLM-enabled simulations, our interviews with experts revealed that there is a high enthusiasm for using such tools in the ESM domain, especially in collaboration with junior developers who lack knowledge about human behavior simulation but are familiar with programming, aligning with the growing interest (e.g., [8, 24, 28, 50] and [14]) and potential of participant behavior simulation in HCI research. Even though we automatically augment the input prompts, the user must be able to specify the right prompts, understand and test the code, and its quality, and spot flaws; calling for collaboration with non-technical domain experts and technical experts when using LLM-based code generators. LLMs may reshape future software development teams, but research and development are required to address the existing concerns and harness the full potential of LLM-based code generation in the software engineering domain, e.g., by reducing

the required technical expert knowledge by autonomously checking the generated code by other LLMs and iterating for a better quality code before outputting its results to the end user.

We see significant value in sharing our journey of training an LLM, to address the substantial challenge of overcoming data deficiency in the ESM domain. Simulating human response behavior and generating user data is shown to be a promising direction and has the potential to advance further by using LLMs. Such endeavors serve to compensate for the scarcity of open-access ESM data and mitigate recruitment impediments. The dissemination of our insights holds the potential to benefit researchers operating within this domain. To this end, we have open-sourced Prompt4Code and reported all our prompt design alternatives along with their output thereby offering a valuable benchmark for the future of LLM applications.

ACKNOWLEDGMENTS

This project was financed by the Dutch Research Council (NWO), grant number 628.011.214.

REFERENCES

[1] 2023. Introducing ChatGPT. Online. https://openai.com/blog/chatgpt Accessed on 2023-07-05.

[2] Aakash Ahmad, Muhammad Waseem, Peng Liang, Mahdi Fehmideh, Mst Shamima Aktar, and Tommi Mikkonen. 2023. Towards Human-Bot Collaborative Software Architecting with ChatGPT. https://doi.org/10.48550/arXiv.2302.14600 arXiv:2302.14600 [cs]

[3] A.Khanshan. 2023. *Prompt4Code*. https://github.com/khnshn/prompt4code Accessed on 2023-07-04.

[4] Hussam Alkaissi and Samy I McFarlane. 2023. Artificial hallucinations in ChatGPT: implications in scientific writing. *Cureus* 15, 2 (2023).

[5] Nikolaos Batalas, Ioanna Lykourentzou, Vassilis-Javed Khan, and Panos Markopoulos. 2021. Reconsidering end-user development definitions. In *International Symposium on End User Development*. Springer, 19–35.

[6] Eva Susann Becker, Thomas Goetz, Vinzenz Morger, and John Ranellucci. 2014. The importance of teachers' emotions and instructional behavior for their students' emotions – An experience sampling analysis. *Teaching and Teacher Education* 43 (Oct. 2014), 15–26. https://doi.org/10.1016/j.tate.2014.05.002

[7] Andre Bishay. 1996. Teacher motivation and job satisfaction: A study employing the experience sampling method. *Journal of undergraduate Sciences* 3, 3 (1996), 147–155.

[8] Paul Brie, Nicolas Burny, Arthur Sluÿters, and Jean Vanderdonckt. 2023. Evaluating a large language model on searching for gui layouts. *Proceedings of the ACM on Human-Computer Interaction* 7, EICS (2023), 1–37.

[9] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[10] Amy Bruckman and Elizabeth Edwards. 1999. Should we leverage natural-language knowledge? An analysis of user errors in a natural-language-style programming language. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*. 207–214.

[11] Yuzhe Cai, Shaoguang Mao, Wenshan Wu, Zehua Wang, Yaobo Liang, Tao Ge, Chenfei Wu, Wang You, Ting Song, Yan Xia, Jonathan Tien, and Nan Duan. 2023. Low-code LLM: Visual Programming over LLMs. https://doi.org/10.48550/arXiv.2304.08103 arXiv:2304.08103 [cs]

[12] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2022. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311* (2022).

[13] S. Consolvo and M. Walker. 2003. Using the experience sampling method to evaluate ubicomp applications. *IEEE Pervasive Computing* 2, 2 (April 2003), 24–31. https://doi.org/10.1109/MPRV.2003.1203750 Conference Name: IEEE Pervasive Computing.

[14] Samuel Rhys Cox, Ashraf Abdul, and Wei Tsang Ooi. 2023. Prompting a large language model to generate diverse motivational messages: A comparison with human-written messages. In *Proceedings of the 11th International Conference on Human-Agent Interaction*. 378–380.

[15] Mihaly Csikszentmihalyi and Reed Larson. 2014. Validity and Reliability of the Experience-Sampling Method. In *Flow and the Foundations of Positive Psychology: The Collected Works of Mihaly Csikszentmihalyi*, Mihaly Csikszentmihalyi (Ed.). Springer Netherlands, Dordrecht, 35–54. https://doi.org/10.1007/978-94-017-9088-8_3

[16] Yihong Dong, Xue Jiang, Zhi Jin, and Ge Li. 2023. Self-collaboration Code Generation via ChatGPT. https://doi.org/10.48550/arXiv.2304.07590 arXiv:2304.07590 [cs]

[17] Gudrun Eisele, Hugo Vachon, Ginette Lafit, Peter Kuppens, Marlies Houben, Inez Myin-Germeys, and Wolfgang Viechtbauer. 2022. The Effects of Sampling Frequency and Questionnaire Length on Perceived Burden, Compliance, and Careless Responding in Experience Sampling Data in a Student Population. *Assessment* 29, 2 (March 2022), 136–151. https://doi.org/10.1177/1073191120957102 Publisher: SAGE Publications Inc.

[18] Common Weakness Enumeration. 2023. *CWE Definitions list and vulnerabilities for CWE entries.* https://www.cvedetails.com/cwe-definitions.php Accessed on 2023-07-03.

[19] James Finnie-Ansley, Paul Denny, Brett A. Becker, Andrew Luxton-Reilly, and James Prather. 2022. The Robots Are Coming: Exploring the Implications of OpenAI Codex on Introductory Programming. In *Proceedings of the 24th Australasian Computing Education Conference* (New York, NY, USA, 2022-02-14) *(ACE '22)*. Association for Computing Machinery, 10–19. https://doi.org/10.1145/3511861.3511863

[20] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. 2024. Retrieval-Augmented Generation for Large Language Models: A Survey. arXiv:2312.10997 [cs.CL]

[21] Nigel Gilbert and Klaus Troitzsch. 2005. *Simulation for the social scientist.* McGraw-Hill Education (UK).

[22] GitHub and OpenAI. 2023. *GitHub Copilot · Your AI pair programmer.* https://github.com/features/copilot Accessed on 2023-06-28.

[23] Rúben Gouveia and Evangelos Karapanos. 2013. Footprint tracker: supporting diary studies with lifelogging. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*. Association for Computing Machinery, New York, NY, USA, 2921–2930. https://doi.org/10.1145/2470654.2481405

[24] Perttu Hämäläinen, Mikke Tavast, and Anton Kunnari. 2023. Evaluating large language models in generating synthetic hci research data: a case study. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–19.

[25] Joel M. Hektner, Jennifer A. Schmidt, and Mihaly Csikszentmihalyi. 2007. *Experience Sampling Method: Measuring the Quality of Everyday Life.* SAGE, Thousand Oaks, California, USA. Google-Books-ID: gLV38Bzf0XQC.

[26] Eric J. Horvitz, Andy Jacobs, and David Hovel. 2013. Attention-Sensitive Alerting. https://doi.org/10.48550/arXiv.1301.6707 arXiv:1301.6707 [cs].

[27] Stephen S. Intille, John Rondoni, Charles Kukla, Isabel Ancona, and Ling Bao. 2003. A context-aware experience sampling tool. In *CHI '03 Extended Abstracts on Human Factors in Computing Systems (CHI EA '03)*. Association for Computing Machinery, New York, NY, USA, 972–973. https://doi.org/10.1145/765891.766101

[28] Bernard J. Jansen, Soon gyo Jung, and Joni Salminen. 2023. Employing large language models in survey research. *Natural Language Processing Journal* 4 (2023), 100020. https://doi.org/10.1016/j.nlp.2023.100020

[29] Melanie M Keller, Mei-Lin Chang, Eva S Becker, Thomas Goetz, and Anne C Frenzel. 2014. Teachers' emotional experiences and exhaustion as predictors of emotional labor in the classroom: An experience sampling study. *Frontiers in psychology* 5 (2014), 1442.

[30] Melanie M Keller, Anne C Frenzel, Thomas Goetz, Reinhard Pekrun, and Lauren Hensley. 2014. Exploring teacher emotions: A literature review and an experience sampling study. *Teacher motivation* (2014), 69–82.

[31] Alireza Khanshan. 2023. *SAPPHIRE.* https://github.com/khnshn/SAPPHIRE Accessed on 10-07-2023.

[32] Alireza Khanshan, Pieter Van Gorp, and Panos Markopoulos. 2022. Experiencer: An Open-Source Context-Sensitive Wearable Experience Sampling Tool. In *International Conference on Pervasive Computing Technologies for Healthcare*. Springer, 315–331.

[33] Alireza Khanshan, Pieter Van Gorp, and Panos Markopoulos. 2023. Simulating Participant Behavior in Experience Sampling Method Research. In *Extended Abstracts of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–7.

[34] Raphaël Khoury, Anderson R. Avila, Jacob Brunelle, and Baba Mamadou Camara. 2023. How Secure is Code Generated by ChatGPT? https://doi.org/10.48550/arXiv.2304.09655 arXiv:2304.09655 [cs]

[35] Tae Soo Kim, DaEun Choi, Yoonseo Choi, and Juho Kim. 2022. Stylette: Styling the Web with Natural Language. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems* (New Orleans, LA, USA) *(CHI '22)*. Association for Computing Machinery, New York, NY, USA, Article 5, 17 pages. https://doi.org/10.1145/3491102.3501931

[36] Reed Larson and Mihaly Csikszentmihalyi. 2014. The Experience Sampling Method. In *Flow and the Foundations of Positive Psychology: The Collected Works of Mihaly Csikszentmihalyi*, Mihaly Csikszentmihalyi (Ed.). Springer Netherlands, Dordrecht, 21–34. https://doi.org/10.1007/978-94-017-9088-8_2

[37] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 9459–9474. https://proceedings.neurips.cc/paper_files/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf

[38] Amanda Li, Jason Wu, and Jeffrey P Bigham. 2023. Using LLMs to Customize the UI of Webpages. In *Adjunct Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology* (San Francisco, CA, USA) *(UIST '23*

Adjunct). Association for Computing Machinery, New York, NY, USA, Article 45, 3 pages. https://doi.org/10.1145/3586182.3616671

[39] Peng Liao, Kristjan Greenewald, Predrag Klasnja, and Susan Murphy. 2020. Personalized HeartSteps: A Reinforcement Learning Algorithm for Optimizing Physical Activity. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 4, 1 (March 2020), 18:1–18:22. https://doi.org/10.1145/3381007

[40] Greg Little and Robert C Miller. 2006. Translating keyword commands into executable code. In *Proceedings of the 19th annual ACM symposium on User interface software and technology*. 135–144.

[41] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023. Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation. https://doi.org/10.48550/arXiv.2305.01210 arXiv:2305.01210 [cs]

[42] Robert Logan IV, Ivana Balazevic, Eric Wallace, Fabio Petroni, Sameer Singh, and Sebastian Riedel. 2022. Cutting Down on Prompts and Parameters: Simple Few-Shot Learning with Language Models. In *Findings of the Association for Computational Linguistics: ACL 2022*. Association for Computational Linguistics, Dublin, Ireland, 2824–2835. https://doi.org/10.18653/v1/2022.findings-acl.222

[43] Mazlina A. Majid, Mohammed Fakhreldin, and Kamal Z. Zuhairi. 2016. Comparing Discrete Event and Agent Based Simulation in Modelling Human Behaviour at Airport Check-in Counter. In *Human-Computer Interaction. Theory, Design, Development and Practice*, Masaaki Kurosu (Ed.). Springer International Publishing, Cham, 510–522.

[44] Klaus G. Müller and Tony Vignaux. 2002. *Overview — SimPy 4.0.2.dev1+g2973dbe documentation.* https://simpy.readthedocs.io/en/latest/ Accessed on 2023-07-04.

[45] Inbal Nahum-Shani, Shawna N Smith, Bonnie J Spring, Linda M Collins, Katie Witkiewitz, Ambuj Tewari, and Susan A Murphy. 2018. Just-in-Time Adaptive Interventions (JITAIs) in Mobile Health: Key Components and Design Principles for Ongoing Health Behavior Support. *Annals of Behavioral Medicine* 52, 6 (May 2018), 446–462. https://doi.org/10.1007/s12160-016-9830-8

[46] Felix Naughton, Muhammad Riaz, and Stephen Sutton. 2016. Response Parameters for SMS Text Message Assessments Among Pregnant and General Smokers Participating in SMS Cessation Trials. *Nicotine & Tobacco Research* 18, 5 (May 2016), 1210–1214. https://doi.org/10.1093/ntr/ntv266

[47] OpenAI. 2023. *Chatgpt.* https://openai.com/chatgpt

[48] OpenAI. 2023. GPT-4 Technical Report. https://doi.org/10.48550/arXiv.2303.08774 arXiv:2303.08774 [cs]

[49] OpenAI. 2023. OpenAI API. Online. https://platform.openai.com Accessed on 2023-06-12.

[50] Joon Sung Park, Lindsay Popowski, Carrie Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2022. Social Simulacra: Creating Populated Prototypes for Social Computing Systems. In *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology* (Bend, OR, USA) *(UIST '22)*. Association for Computing Machinery, New York, NY, USA, Article 74, 18 pages. https://doi.org/10.1145/3526113.3545616

[51] Fabio Paternò and Volker Wulf. 2017. *New perspectives in end-user development.* Springer.

[52] Baolin Peng, Michel Galley, Pengcheng He, Hao Cheng, Yujia Xie, Yu Hu, Qiuyuan Huang, Lars Liden, Zhou Yu, Weizhu Chen, and Jianfeng Gao. 2023. Check Your Facts and Try Again: Improving Large Language Models with External Knowledge and Automated Feedback. https://doi.org/10.48550/arXiv.2302.12813 arXiv:2302.12813 [cs]

[53] Steven I. Ross, Fernando Martinez, Stephanie Houde, Michael Muller, and Justin D. Weisz. 2023. The Programmer's Assistant: Conversational Interaction with a Large Language Model for Software Development. In *Proceedings of the 28th International Conference on Intelligent User Interfaces* (New York, NY, USA, 2023-03-27) *(IUI '23)*. Association for Computing Machinery, 491–514. https://doi.org/10.1145/3581641.3584037

[54] Ohad Rubin, Jonathan Herzig, and Jonathan Berant. 2022. Learning To Retrieve Prompts for In-Context Learning. https://doi.org/10.48550/arXiv.2112.08633 arXiv:2112.08633 [cs]

[55] Murray Shanahan. 2022. Talking About Large Language Models. *arXiv preprint arXiv:2212.03551* (2022).

[56] Jasmine Shone and David Kim. 2022. Yes, You Can Make an App Too: A Systematic Study of Prompt Engineering in the Automatic Generation of Mobile Applications from User Queries. (2022).

[57] Arthur A. Stone, Ronald C. Kessler, and Jennifer A. Haythomthwatte. 1991. Measuring Daily Events and Experiences: Decisions for the Researcher. *Journal of Personality* 59, 3 (1991), 575–607. https://doi.org/10.1111/j.1467-6494.1991.tb00260.x _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-6494.1991.tb00260.x.

[58] Arthur A. Stone and Saul Shiffman. 1994. Ecological momentary assessment (EMA) in behavorial medicine. *Annals of Behavioral Medicine* 16 (1994), 199–202. https://doi.org/10.1093/abm/16.3.199 Place: US Publisher: Lawrence Erlbaum.

[59] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning, second edition: An Introduction.* MIT Press, Cambridge, Massachusetts, USA. Google-Books-ID: uWV0DwAAQBAJ.

[60] Niels van Berkel, Denzil Ferreira, and Vassilis Kostakos. 2017. The Experience Sampling Method on Mobile Devices. *ACM Comput. Surv.* 50, 6, Article 93 (dec 2017), 40 pages. https://doi.org/10.1145/3123988

[61] Niels van Berkel, Jorge Goncalves, Lauri Lovén, Denzil Ferreira, Simo Hosio, and Vassilis Kostakos. 2019. Effect of experience sampling schedules on response rate and recall accuracy of objective self-reports. *International Journal of*

Human-Computer Studies 125 (May 2019), 118–128. https://doi.org/10.1016/j.ijhcs.2018.12.002

[62] Shihan Wang, Karlijn Sporrel, Herke van Hoof, Monique Simons, Rémi D. D. de Boer, Dick Ettema, Nicky Nibbeling, Marije Deutekom, and Ben Kröse. 2021. Reinforcement Learning to Send Reminders at Right Moments in Smartphone Exercise Application: A Feasibility Study. *International Journal of Environmental Research and Public Health* 18, 11 (Jan. 2021), 6059. https://doi.org/10.3390/ijerph18116059 Number: 11 Publisher: Multidisciplinary Digital Publishing Institute.

[63] Shihan Wang, Chao Zhang, Ben Kröse, and Herke van Hoof. 2021. Optimizing Adaptive Notifications in Mobile Health Interventions Systems: Reinforcement Learning from a Data-driven Behavioral Simulator. *Journal of Medical Systems* 45, 12 (Oct. 2021), 102. https://doi.org/10.1007/s10916-021-01773-0

[64] Jules White, Quchen Fu, Sam Hays, Michael Sandborn, Carlos Olea, Henry Gilbert, Ashraf Elnashar, Jesse Spencer-Smith, and Douglas C. Schmidt. 2023. A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT. https://doi.org/10.48550/arXiv.2302.11382 arXiv:2302.11382 [cs]

[65] Zhenchang Xing, Qing Huang, Yu Cheng, Liming Zhu, Qinghua Lu, and Xiwei Xu. 2023. Prompt Sapper: LLM-Empowered Software Engineering Infrastructure for AI-Native Services. https://doi.org/10.48550/arXiv.2306.02230 arXiv:2306.02230 [cs]

[66] Chao Zhang, Shihan Wang, Henk Aarts, and Mehdi Dastani. 2021. Using Cognitive Models to Train Warm Start Reinforcement Learning Agents for Human-Computer Interactions. https://doi.org/10.48550/arXiv.2103.06160 arXiv:2103.06160 [cs].

[67] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. 2023. A Survey of Large Language Models. https://doi.org/10.48550/arXiv.2303.18223 arXiv:2303.18223 [cs]

Table 1. Details of all prompting iterations and strategies explored in this paper.

#	Code version	Prompt	Prompting strategy	Meding action(s)	Positive remarks	Explanation of the error(s)	Syntax error	Semantic correctness
1	1	A	Zero-shot	n/a	The code uses SimPy	Includes inexecutable strings in the beginning and end of the file ```python ... ```	✓	✗
2	1.1	n/a	n/a	Manually removed inexecutable string in the 1st iteration	The code executes	- Does not output the response rate - The response rate is randomly generated	✗	✗
3	2	A	Zero-shot	n/a	The code uses SimPy	'None is not a generator' error	✓	✗
4	2.1	n/a	n/a	Asked ChatGPT to fix the bug	The code uses SimPy	- ChatGPT failed to find any bugs - Throws None is not a generator error	✓	✗
5	2.2	n/a	n/a	Pointed ChatGPT to what and where the bug is then asked to fix it	- The code uses SimPy - The code executes	- The code does not output the response rate - The simulation runs for less than 50 teachers	✗	✗
6	3	A	Zero-shot	n/a	- The code uses SimPy - The code executes	- Simulates teachers with even index - The code does not calculate the response rate	✗	✗
7	4	A	One-shot	Prompted B as a system message and its corresponding handcrafted code for in-context learning	The code uses SimPy	- Name 'Enum' is not defined error - Runtime error in using interruptions	✓	✗
8	4.1	n/a	n/a	Asked ChatGPT to solve the bug	- The code uses SimPy - The code executes	The response rate is calculated purely randomly	✗	✗
9	5	A	One-shot	n/a	The code uses SimPy	The 'Timeout' object has no attribute 'response_counter' error	✓	✗
10	5.1	n/a	n/a	Asked ChatGPT to fix the bug	- The code uses SimPy - The code executes - outputs response rate	- Does not use interruptions given as an example during in-context learning - The response rate is randomly generated	✗	✗
11	6	D	One-shot	Prompted C as a system message and its corresponding handcrafted code for in-context learning	Well-formatted JSON configuration	The decision body property was not in the right format	✗	✗
12	6.1	E	~	n/a	Well-formatted JSON configuration	The decision body is executable but does not comply with the function structure	✗	✗
13	6.2	F	~	n/a	The code executes as expected	n/a	✗	✓
14	7	G	Few-shot	Passed multiple (in our case 7 simulation source codes) examples as system messages for in-context learning	The code executes as expected	n/a	✗	✓
15	8	H	~	Crafted a more complex prompt	The code executes	The calculations of domain-specific theories are too simplistic and random	✗	✗

Table 2. Prompts used in iterations of Table 1.

Letter	Prompt
A	Write a Python program using the SimPy framework that simulates teacher's activities during the day. Teachers will receive notifications on their smartphones every 2 hours. Then depending on their situation they either respond to the notification or ignore it. The program should estimate the response rate to such notifications for 50 teachers.
B	Write a Python program using the SimPy framework that simulates teacher's activities during the day. Teachers will receive notifications on their smartphones at random moments. Then depending on their situation they either respond to the notification or ignore it. The program should simulate a single teacher in 1000 timesteps.
C	Write a configuration dictionary in Python that specifies the execution of a simulation that runs for 1000 ticks, simulates a teacher, a teacher self-reports, and takes a random time between 2 to 5 ticks. Teachers receive beeps randomly between every 50 to 100 ticks. The decision body refers to what a teacher takes into account before engaging (or not) with a beep. In this case, it should always be True. A teacher has two routines, teaching and taking a break. While teaching, a teacher is busy with a 90% chance, and the teaching duration is fixed for 150 ticks. A teacher is busy with a 10% chance when taking a break. The duration of a break is random between 5 to 10 ticks.
D	Write a configuration dictionary in Python that specifies the execution of a simulation that runs for 4000 ticks, simulates a teacher, a teacher self-reports, and takes a random time between 1 to 4 ticks. Teachers receive beeps randomly between every 60 to 150 ticks. The decision body should be random, sometimes true, other times false. A teacher has four routines, teaching, walking to the break room, taking a break, and walking back to the classroom. The duration of each activity and how busy a teacher is during each activity are random.
E	Same as D but provided more clarity of the format of the decision body: …the decision body must be an executable python…
F	Same as E but further clarified the decision body: … must set the value of self.i_want_to to either true or false…
G	Write a Python program using SimPy that simulates teacher's activities during the day. Teachers will receive notifications on their smartphones every 2 hours. Then depending on their situation they either respond to the notification or ignore it. Estimate the response rate to such notifications. Run such a program for 50 teachers. Output executable code with no comments and no explanations.
H	Write a Python program using SimPy that simulates teachers' activities during the day. Teachers will receive notifications on their smartphones every 2 hours. Then depending on their situation they either respond to the notification or ignore it. To better approximate the response behavior of teachers, take into consideration the attentional availability of the teachers and their level of motivation. The attentional availability should be calculated with the expected cost of interruption versus the benefit of giving a response. The level of motivation should be a linear function of time. For example, motivation levels might decrease at the end of the day. Accordingly, estimate the response rate to such notification. Run such a program for 50 teachers.

First Submitted October 2023; Revised February 2024; Revised April 2024; Accepted April 2024