Transforming Ideas into Code: Visual Sketching for ML Development

Luís Gomes
lfgomes@andrew.cmu.edu
Carnegie Mellon University
Pittsburgh, PA, USA
University of Porto
Porto, Portugal

Abstract

We propose a novel code assistant and generation paradigm aimed at closing the gap between visual sketching and code creation for Machine Learning (ML) development. This approach empowers developers and ML practitioners to translate hand-drawn sketches into functional code with enhanced accuracy and usability. Developers are recruited to assess the tool's performance. This research contributes to the future of low-code approaches, facilitating ML application development, and promoting an intuitive and accessible programming environment.

CCS Concepts: • **Software and its engineering** → *Visual languages*; **Automatic programming**; • **Computing methodologies** → **Computer vision**.

Keywords: Code generation, Visual sketching, Machine learning, Tool development, Synthetic data

ACM Reference Format:
Luís Gomes. 2023. Transforming Ideas into Code: Visual Sketching for ML Development. In *Companion Proceedings of the 2023 ACM SIGPLAN International Conference on Systems, Programming, Languages, and Applications: Software for Humanity (SPLASH Companion '23), October 22–27, 2023, Cascais, Portugal.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3618305.3623588

1 Introduction

Writing code is challenging, time-consuming, and repetitive. The use of low-code approaches can facilitate Machine Learning (ML) application development for individuals with domain expertise but limited programming reasoning [9]. On the other hand, experienced software developers heavily rely on sketches and diagrams to externalize their mental models, share ideas, and aid in code implementation [2, 10]. Manually translating sketches into code is error-prone, and lacks traceability, leading to discarded initial drawings [4].

1.1 Problem

Software developers rely on sketches and diagrams in their daily work [2]. The link between the final implementation and the original sketch is often lost after erasing the sketch and there is a need to implement it into the code [3, 4].

Although code generation tools have been recently released to help programmers write their code, generating it from documentation or text descriptions, it's not easy to provide them with the spatial information that is represented in diagrammatic sketches, such as system architectures or Data Science workflows. Furthermore, novice scientists or students from many fields who want to start programming highly benefit from low-code and visual representations integrated into the process [15]. These tasks can be automated, linking code artifacts to sketches and converting the information from informal hand-drawn sketches into usable code [4, 7]. My collaborators and I hypothesize that code assistants capturing visual information and reusing sketches by developers will greatly benefit software development.

1.2 Related Work

As shown in Figure 1, this project intends to work from different lines of research. From a technical perspective (blue), we intend to leverage the advancements in *Computer Vision*, using segmentation and classification models [11], make use of Large Language Models *LLMs* to generate code [12, 16], and explore techniques used in similar fields (*Img2Code*) where images are used to generate latex or HTML code [8, 13]. From a human perspective (orange), we want to understand the impact of *Sketching* [1, 5, 6] in ML scenarios, how it relates to *AI Understanding* and how low-code platforms (*Visual Low-Code*) facilitate building applications [9].

There is a gap between *Sketching* and generative code research (*Img2Code*, *Language Models*). D'Amorim *et al.* [7] proposes the automatic code generation of Data Science concepts. Our aim is to create a pioneering code generation tool that bridges visual sketching and efficient code creation. By applying their idea to full ML workflows, we explore a new research direction that combines vision tools and LLMs to

SPLASH Companion '23, October 22–27, 2023, Cascais, Portugal

ACM ISBN 979-8-4007-0384-3/23/10.
https://doi.org/10.1145/3618305.3623588

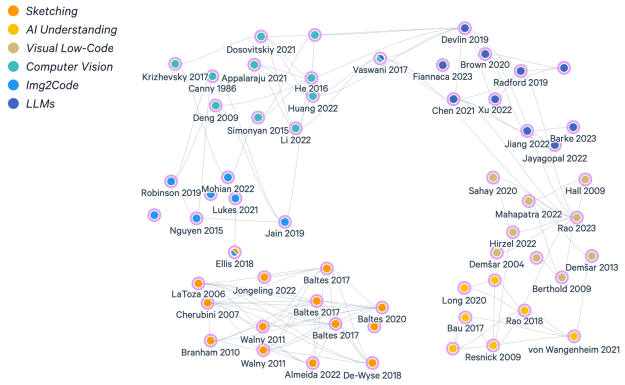

Figure 1. Related Work - Literature map.

generate code, mapping it directly to developers' mental models. Successfully implementing this idea in ML tasks, opens the door to use a similar approach in other areas, expanding the visual sketching paradigm to broader SE domains. Bridging these lines of work, we suggest creating a visual sketching code assistant with sketch-to-code tracking, auto code generation, and a drag-and-drop low-code interface to connect sketching and coding. To understand these issues, we iteratively prototype a tool to convert sketches into code, refining it based on user feedback to meet developer needs effectively. Compared to other works, that focus either only on code generation or singular benefits of sketch representations, we intend to create a tool that takes advantage of both, always grounded on what the user needs.

2 Motivation

As presented by many authors, informal sketches play an important role in shaping software products, helping developers, especially in 2 moments: *a)* the early planning phases as externalized mental models and *b)* later as documentation for the code. Reusing those sketches will improve the developers' productivity by *a)* reducing the time they need to code their ideas and *b)* reducing the time on understanding the code. We will focus on Data Science and ML applications. The need to build systems with AI has been growing and it makes sense to explore the problem by directing the attention to this sub-field of Software Engineering (SE).

2.1 Importance

The research on converting hand-drawn sketches to code holds immense significance in modern software development and ML. By introducing Sketch Programming assistants, an intuitive paradigm based on visual sketching, this research advances Human-Computer Interaction (HCI) and facilitates software development. Embracing sketches as a universal language bridges communication gaps, enabling cross-domain collaboration and empowering novice developers to engage with coding more easily. Moreover, the implementation of a visual sketching code assistant fosters innovation in visual programming, facilitates user-centric development, and contributes to broader SE research.

2.2 Difficulties

One of the obstacles is employing Computer vision (CV) to accurately read and interpret hand-drawn sketches. This process requires robust algorithms to identify elements and reconstruct diagrams in a clean manner, essential for generating good code. Data scarcity poses another challenge. Training CV models necessitates a large and diverse dataset of hand-drawn sketches, which can be challenging to obtain and annotate effectively. Furthermore, creating a Domain Specific Language (DSL) that captures both visual information and code semantics is crucial. The DSL must bridge the gap between the two domains, balancing simplicity and expressiveness. Additional difficulties include handling ambiguity in hand-drawn notations, ensuring real-time conversion capabilities, managing complex diagrams, and implementing error handling and validation effectively.

2.3 Impact

The successful conversion of hand-drawn sketches to code bears significant implications for programming. We hope to introduce a new programming assistant paradigm, Sketch Programming, representing a departure from the traditional Natural Language to code approaches. By incorporating visual sketching as a means of code creation, Sketch Programming offers an intuitive alternative that could revolutionize how developers interact with code. Additionally, we offer valuable insights into how ML developers and data scientists use sketches to express their intent.

Understanding these practices can inform the design of more effective tools, streamlining the development process and improving coding efficiency and accuracy. The successful implementation of the tool will enhance code generation convenience, transparency, and ease of review and modification.

3 Approach

Our approach is based on iterative prototyping. We first build a prototype of a tool that generates code from a hand-drawn sketch. Even with limited accuracy, the current version is able to read an image, segment it into parts, and identify symbols, such as plots or neural networks, creating a Jupyter notebook with the corresponding code cells. Arrows and text are also identified. The information is captured by our DSL and the code is generated by an LLM using the text and description of each symbol. Each version of the tool is presented to users and it is modified based on their feedback. It allows us to detail and document the user needs, improving the tool over time. Interviews and surveys are also being conducted, asking questions about the planning

process, and observing the participants sketching. The users are data scientists with different expertise levels, who want to design data workflows, maintain their models documented, and minimize programming efforts. Examining their form of sketching is another goal of the studies.

Therefore, we advance in 3 dimensions: advance technology by creating and improving the tool, gain new knowledge about user needs in the context of sketching, and introduce a novel paradigm for programming assistants.

4 Evaluation Methodology

The main hypothesis states that creating a code generation tool from hand-drawn sketches is not only possible but also has the potential to significantly enhance the productivity of ML developers. There are two secondary hypotheses. The first one suggests that using the code generation tool improves code quality. By incorporating a linking mechanism, users are encouraged to understand the code generated from their sketches and link it to the underlying concepts. This is expected to reduce bugs introduced on generation, since the overconfidence in AI-generated code [14] is compensated by increased awareness during development.

To overcome data availability issues, we suggest generating synthetic data comparable to human-drawn sketches. This other secondary hypothesis states that synthetic sketches can accurately represent the real ones. We need to test this hypothesis empirically to train the ML models, enhancing their accuracy.

4.1 Evaluation Setup

To study productivity, quantitative variables, such as the time to complete a task, are measured but also qualitative variables that allow us to infer long-term productivity. For instance, the degree of code understanding is important. Developers who understand the code in a project resume programming tasks more easily. We hypothesize that seeing the sketch helps recall the code in future development iterations, losing less time to understand the code. To study code quality we need to study quantitatively the capacity of the tool to generate bug-free code. After the generation step, we also need to test the user's code understanding, introducing bugs in the generated code and testing their capacity to identify and fix them. To understand if data is accurately generated, real-world sketches need to be collected. By creating 2 test sets, one with synthetic data and another with real data, we will be able to compare the ML model metrics for each case.

To control for bias, ML developers from various backgrounds, including students, researchers, and data scientists, are recruited as participants. The sample size is subsequently enlarged over time in an iterative manner for a comprehensive study. To ensure external validity, the performance of our code assistant is tested in real-world scenarios, both in industrial and academic settings.

5 Conclusion

The impact of this research extends beyond technical advancements. It introduces a new way of programming assistance, empowers ML developers with efficient tools, and paves the way for a more intuitive and inclusive programming ecosystem. By leveraging the power of visual sketching, this approach has the potential to reshape the landscape of SE and promote innovative approaches to coding.

References

[1] E. Almeida, Iftekhar Ahmed, and A. Hoek. 2022. Let's Go to the Whiteboard (Again): Perceptions from Software Architects on Whiteboard Architecture Meetings. https://doi.org/10.48550/ARXIV.2210.16089

[2] Sebastian Baltes and Stephan Diehl. 2017. Sketches and Diagrams in Practice. https://doi.org/10.1145/2635868.2635891

[3] Sebastian Baltes, Fabrice Hollerich, and Stephan Diehl. 2017. Round-Trip Sketches: Supporting the Lifecycle of Software Development Sketches from Analog to Digital and Back. https://doi.org/10.1109/VISSOFT.2017.24

[4] Sebastian Baltes, Peter Schmitz, and Stephan Diehl. 2017. Linking Sketches and Diagrams to Source Code Artifacts. https://doi.org/10.1145/2635868.2661672

[5] Stacy M. Branham, G. Golovchinsky, S. Carter, and Jacob T. Biehl. 2010. Let's go from the whiteboard: supporting transitions in work through whiteboard capture and reuse. https://doi.org/10.1145/1753326.1753338

[6] M. Cherubini, Gina Venolia, R. DeLine, and Amy J. Ko. 2007. Let's go to the whiteboard: how and why software developers use drawings. https://doi.org/10.1145/1240624.1240714

[7] Marcelo d'Amorim, Rui Abreu, and Carlos A. B. Mello. 2020. Visual sketching: from image sketches to code. https://doi.org/10.1145/3377816.3381745

[8] Kevin Ellis, Daniel Ritchie, Armando Solar-Lezama, and Joshua B. Tenenbaum. 2018. Learning to Infer Graphics Programs from Hand-Drawn Images.

[9] Martin Hirzel. 2022. Low-Code Programming Models. https://doi.org/10.48550/ARXIV.2205.02282

[10] Thomas D. LaToza, Gina Venolia, and R. DeLine. 2006. Maintaining mental models: a study of developer work habits. https://doi.org/10.1145/1134285.1134355

[11] Junlong Li, Yiheng Xu, Tengchao Lv, Lei Cui, Chaoxi Zhang, and Furu Wei. 2022. DiT: Self-supervised Pre-training for Document Image Transformer. https://doi.org/10.1145/3503161.3547911

[12] OpenAI. 2023. GPT-4 Technical Report. https://doi.org/10.48550/ARXIV.2303.08774

[13] Dipti Pawade, Avani Sakhapara, Sanyogita Parab, Divya Raikar, Ruchita Bhojane, and Henali Mamania. 2018. Automatic HTML Code Generation from Graphical User Interface Image. https://doi.org/10.1109/RTEICT42901.2018.9012284

[14] Neil Perry, Megha Srivastava, Deepak Kumar, and D. Boneh. 2022. Do Users Write More Insecure Code with AI Assistants? https://doi.org/10.48550/ARXIV.2211.03622

[15] A. Rao, Ayush Bihani, and Mydhili K. Nair. 2018. Milo: A visual programming environment for Data Science Education. https://doi.org/10.1109/VLHCC.2018.8506504

[16] Frank F. Xu, Uri Alon, Graham Neubig, and V. Hellendoorn. 2022. A systematic evaluation of large language models of code. https://doi.org/10.1145/3520312.3534862

Received 2023-07-21; accepted 2023-08-10