UNIVERSITY OF CALIFORNIA SAN DIEGO

Synthesizing Transparent and Inspectable Technical Workflows

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy

in

Cognitive Science

by

Ian Zachariah Drosos

2022

DEDICATION

To Angela, this never would have happened without you.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I'm extremely grateful to Philip for being my advisor and providing support and mentorship while getting through the PhD grind. You have made me a better researcher, writer, and teacher. Thank you for giving me the freedom and encouragement to explore the directions that meant the most to me. Thank you to the rest of my committee – Jim, Brad, Nadia, and Ranjit – for your continued support and feedback on my thesis research.

I'm deeply indebted to Titus Barik and the PROSE team, who gave me invaluable insight and the opportunity to create something I'm really proud of. Special thanks to Chris Parnin who gave me my first shot at research and helped turn a software engineer into a researcher. I am so grateful to all the members of the Design Lab, especially Adam Rule, who introduced me to working with computational notebooks which helped shape my research direction tremendously, Sean and Sam, who have been amazing lab mates and provided their support throughout the years, and my CogSci cohort for their friendship.

Many thanks to the UCSD Cognitive Science Department for the opportunity to experience teaching as both a mentor to students and as a lecturer. Thanks to Marta and Andrea for guiding our 2nd and 3rd year research, and all the CogSci staff who helped me navigate UCSD requirements throughout the years! Thank you to the committees and volunteers who reviewed and gave feedback to our work, which was vital in helping our work be the best it could be. Finally, thank you to my family, who provided their unwavering love and support when I needed it most.

Chapter 3, in full, is a reprint of the material as it appears in the proceedings of the 2020 CHI Conference on Human Factors in Computing Systems as Wrex: A Unified Programming-By-Example Interaction for Synthesizing Readable Code for Data Scientists. Ian Drosos, Titus Barik, Philip Guo, Robert DeLine, and Sumit Gulwani. 2020. The dissertation author was the primary investigator and author of this paper.

Chapter 4, in part, is a reprint of the material as it appears in the proceedings of the 2021 Symposium on Visual Languages and Human-Centric Computing as Streamers Teaching

Programming, Art, and Gaming: Cognitive Apprenticeship, Serendipitous Teachable Moments, and Tacit Expert Knowledge. Ian Drosos and Philip Guo. 2021. The dissertation author was the primary investigator and author of this paper.

Chapter 5, in full, is a reprint of the material as it will appear in the proceedings of the 2022 Conference on Designing Interactive Systems as The Design Space of Livestreaming Equipment Setups: Tradeoffs, Challenges, and Opportunities. Ian Drosos and Philip Guo. 2022. The dissertation author was the primary investigator and author of this material.

VITA

2011	Bachelor of Science in Computer Science, Southern Polytechnic State University
2017	Master of Science in Computer Science, North Carolina State University
2022	Doctor of Philosophy in Cognitive Science, University of California San Diego

PUBLICATIONS

Ian Drosos and Philip Guo. 2022. The Design Space of Livestreaming Equipment Setups: Trade-offs, Challenges, and Opportunities. To appear in Proceedings of the Conference on Designing Interactive Systems (DIS 2022).

Ian Drosos and Philip Guo. 2021. Streamers Teaching Programming, Art, and Gaming: Cognitive Apprenticeship, Serendipitous Teachable Moments, and Tacit Expert Knowledge. In Proceedings of the Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2021).

Ian Drosos, Titus Barik, Philip Guo, Robert DeLine, and Sumit Gulwani. 2020. Wrex: A Unified Programming-By-Example Interaction for Synthesizing Readable Code for Data Scientists. In Proceedings of the Conference on Human Factors in Computing Systems (CHI 2020).

ABSTRACT OF THE DISSERTATION

Synthesizing Transparent and Inspectable Technical Workflows

by

Ian Zachariah Drosos

Doctor of Philosophy in Cognitive Science

University of California San Diego, 2022

Professor Philip J. Guo, Chair

The highly technical and complex workflows found in data science and programming have led to many barriers that hinder even an expert while they wrangle code inside their computational notebooks or write code for their data analysis. Further, when these experts want to share these workflows online and act as content creators, they run into similar barriers to the work being done effectively and efficiently. The goal of this research is to discover and remove barriers and frustrations felt by data scientists and programmers in their daily workflows, including data wrangling and the broadcasting of these workflows online.

My thesis is that being able to see and interact with technical workflows behind the scenes can help novices grow into experts, but existing user-friendly computational systems may hinder growth by obscuring important kinds of complexity.

This dissertation supports my thesis by performing three studies. The first study investigates current issues with data wrangling and tools that assist data scientists in their daily data wrangling workflows, and introduces a unified interaction model based on programming-by-example that generates readable code for a variety of useful data transformations, implemented as a Jupyter notebook extension called Wrex. The second study investigates how technical expert content creators, such as programmers and data scientists, are teaching naturalistic workflows via livestreaming on sites like Twitch and YouTube, and how these might lend themselves to a virtual form of cognitive apprenticeship. Finally, the third study investigates the current state of livestreaming equipment setups and develops a design space defining the range of system setups and the challenges faced by content creators when they must build, run, and maintain these complex systems.

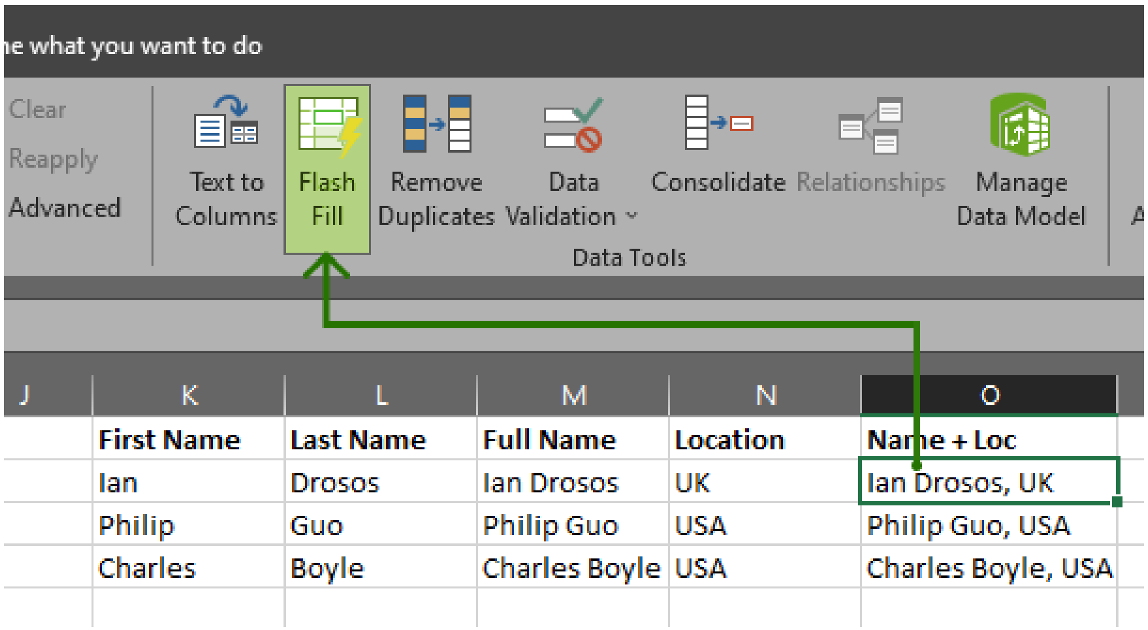
Chapter 1

Introduction

This dissertation explores designing better experiences for experts in cognitively complex domains like data science, programming, and online content creation. First, it investigates the challenges that professional data scientists face when performing data wrangling during their daily data analysis tasks, such as recalling functions required to wrangle their data or context switching between their computational notebook and their data wrangling tools. It then presents a data wrangling tool called Wrex that aims to alleviate many of these challenges for data scientists at work and allow them to share their results with their team. Secondly, it investigates a form of sharing knowledge with novices that I call Instructional Workflow Streams, which may reveal many serendipitous teachable moments and even tacit knowledge that can be cut from the usual focused tutorial video. Finally, this dissertation defines the design space of livestreaming equipment setups to discover challenges faced by livestreaming content creators when creating, maintaining, and running these setups to provide content to their viewers.

1.1 Motivation: challenges in understanding technical workflows

Workflows that require advanced technical skill, like those seen in data analysis and programming, often require difficult and tedious work. Whether a programmer is debugging an issue with their code or a data scientist is performing data wrangling on a new dataset using



First Name	Last Name	Full Name	Location	Name + Loc
Ian	Drosos	Ian Drosos	UK	Ian Drosos, UK
Philip	Guo	Philip Guo	USA	Philip Guo, USA
Charles	Boyle	Charles Boyle	USA	Charles Boyle, USA

Figure 1.1. FlashFill [48] can take data in several different columns and perform data wrangling automatically for a user based on their intention by the providing of one or more input-output examples. In this example, the inputs 'Ian Drosos' and 'UK' and the example output 'Ian Drosos, UK' enabled FlashFill to perform the same transformations in the proceeding rows.

unfamiliar APIS, these experts still frequently have issues with recalling how to use certain functions or fix their current errors. My earlier research on the design space of 60 computational notebooks found that much work has gone into alleviating these issues for programmers and data scientists [82], which have resulted in helpful tools for cleaning data quicker, organizing notebooks better, or synthesizing code to solve the problem itself with programming-by-example, but there are still gaps in usability of these tools. For example, in code synthesis and end-user programming, there is FlashFill that can fill in a column with data based on the input-output examples a user gives (Figure 1.1). There are also AI systems like GitHub Copilot [43] that can provide potentially useful code snippets to programmers, but the problem is that these systems, the interactions with them, and their output can seem like magic.

Some of this is due to a lack of transparency as it is hard for people to interpret what is going on to make these tools work and understand their outputs, especially when things need to be modified. In FlashFill they might not know how to perform this transformation on other datasets or trust the transformation will even work with all of their edge cases. By creating tools that are transparent, users can trust, verify, and further manipulate the results of these tools. Without this transparency, users might be less likely to adopt a new tool or feature into their current workflow as the flexibility of the solution may get called into question. *If a user can understand what the tool is doing, it can make it easier to modify the output of the tool to better align it with the user's preferences or goals.*

There is also a lack of assistance for experts in teaching or sharing their knowledge with others. What if they actually want to teach people how to do these tasks and share this knowledge? What strategies do current experts employ, and what might future tools do to help them? Performing the tasks they normally perform in their daily workflows is difficult enough, now these expert content creators have to also think about how to disseminate the information to their viewers who often are interested in learning the ins and outs of data science or programming, including the tricks of the trade that are not easily explainable due to tacit (unspoken) knowledge. *By making these technical workflows more transparent, learners can gain these insights.*

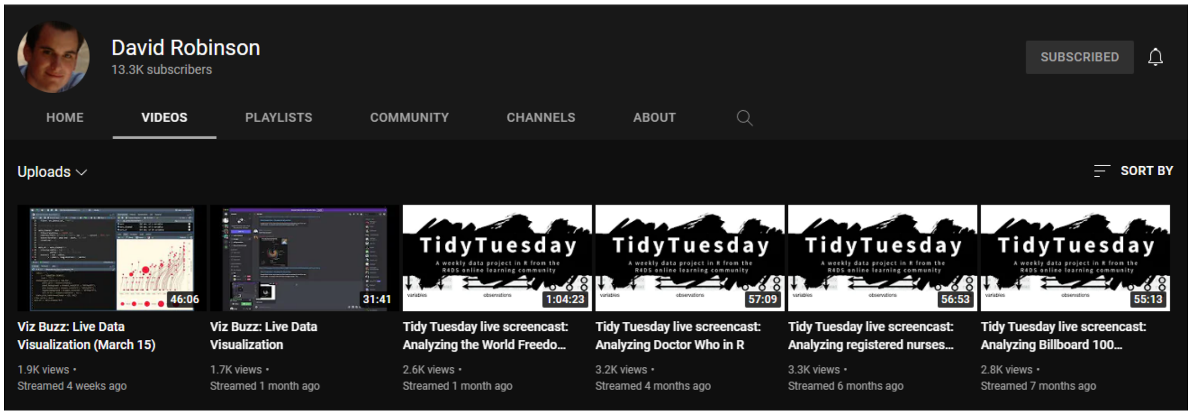

David Robinson
13.3K subscribers
SUBSCRIBED
HOME
VIDEOS
PLAYLISTS
COMMUNITY
CHANNELS
ABOUT
Uploads
SORT BY
46:06
Viz Buzz: Live Data Visualization (March 15)
1.9K views • Streamed 4 weeks ago
31:41
Viz Buzz: Live Data Visualization
1.7K views • Streamed 1 month ago
1:04:23
Tidy Tuesday live screencast: Analyzing the World Freedo...
2.6K views • Streamed 1 month ago
57:09
Tidy Tuesday live screencast: Analyzing Doctor Who in R
3.2K views • Streamed 4 months ago
56:53
Tidy Tuesday live screencast: Analyzing registered nurses...
3.3K views • Streamed 6 months ago
55:13
Tidy Tuesday live screencast: Analyzing Billboard 100...
2.8K views • Streamed 7 months ago

Figure 1.2. Some content creators like David Robinson stream their entire workflows for cognitively demanding and complex tasks like data science. [121]

One popular knowledge-sharing method seeing huge growth online is the streaming of entire technical workflows, in domains like data analysis, programming, digital art, and even gaming as people stream their workflows to their audience (Figure 1.2). While these streams can lack the organization and focus of traditional tutorial videos, there may be valuable content being revealed within these streams that do not make it into targeted tutorial videos due to the challenges faced by content creators when trying to cover all the nuances required to complete tasks found in these workflows. The longer format of these streams (up to a few hours each) reveal daily tedium and other difficulties faced by these content creators that may serve as lessons to novices in the audience who are watching to gain new skills and to perhaps become experts themselves.

But even as promising as this medium is for experts to share the knowledge of their craft with viewers, the acquiring, setting up, modifying, maintaining, and running of the equipment needed in the production of a livestream can also introduce barriers for experts who want to create streaming content. New streamers must balance the sound and video fidelity of their stream within their budget. Producing the stream live, while also working on cognitively demanding tasks, can be helped with hardware and software tools that assist the content creator in becoming a one-person production crew (Figure 1.3), switching between different scenes and effects as relevant to the work they are doing live on stream. To this, there are still many opportunities for new interactions for assisting these users in reaching their goals as a content creator once we better understand the current challenges that content creators face when maintaining their streaming setups.

1.2 Motivating Example

To give a different perspective on these challenges, consider Dan the data scientist who wants to solve some data wrangling tasks for a new data analysis he is working on:

Wrangling workflows: Dan first loads up a dataset for his data analysis in his preferred computational notebook tool for doing data science, Jupyter notebook. Dan's first goal with any data analysis is to first make sure his data is free of any issues that would inhibit his analysis, so Dan begins the process of data wrangling (transforming, shaping, and cleaning data to make it suitable for analysis) with a slice of the dataset. However, Dan runs into a few issues while trying to wrangle his data, namely figuring out what specific code he needs to write to wrangle this

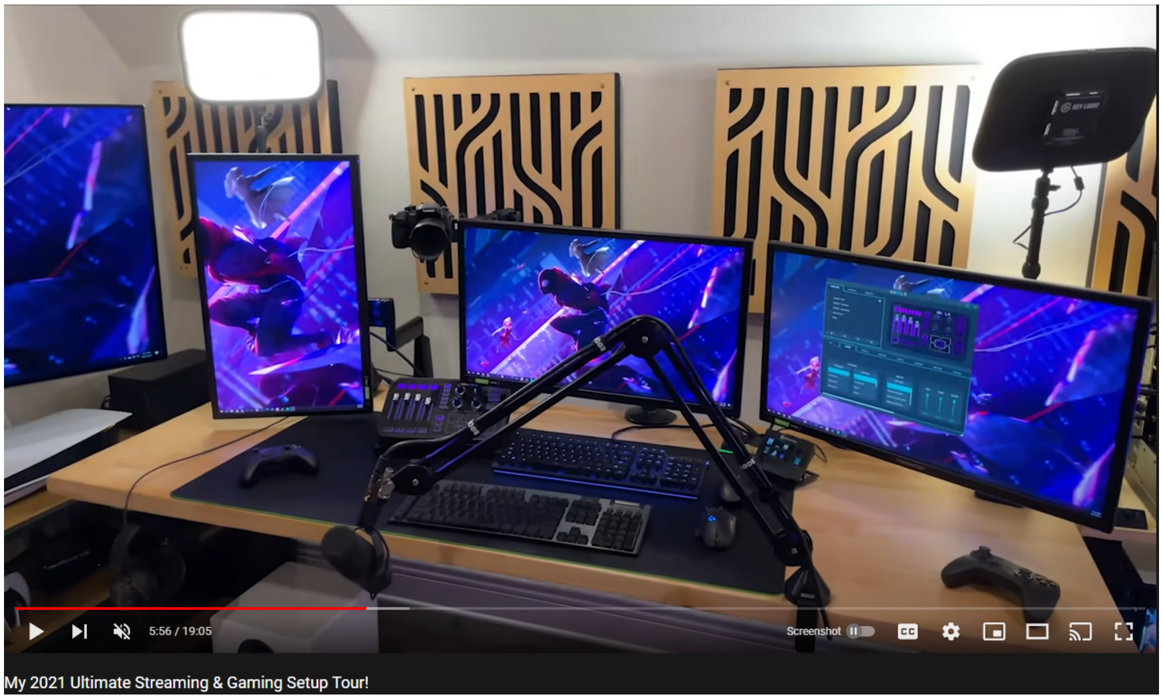


Figure 1.3. An example streaming setup published on YouTube which includes multiple monitors, PCs, mixers, and peripherals, as well as a Streamdeck to assist in the live production of a stream.

data, recalling the functions and regex (regular expressions) required to perform the transforms he desires, and running this code on the entire dataset or other similar datasets. In order to figure this out, Dan goes to search online on websites like StackOverflow for examples of the data transformations he would like to do whenever he runs into recalling what code would be fruitful to try given his task. Dan then searches APIs and other documentation for help and realizes all of this work has removed him from the context of doing the data analysis that he originally wants to do in the first place. Current data wrangling tools simply transform the data without revealing the methods they use to do so. So while Dan's data is starting to shape up, he cannot apply this solution to his full dataset without going though all the steps again manually. Further, while he can inspect the resulting data to attempt to make sure the tool performed his transforms as expected, without the code performing these transforms he is unable to determine if it can handle edge cases that might be found within the complete dataset. Dan does find some use with current tools, but without the ability to read the code performing these transforms, he is unable to trust using them in his daily workflow. Further, Dan is not too happy about having to switch out of his Jupyter notebook to work with external data wrangling tools where he has to export and import data from and to his notebook and would just prefer to just do it all by hand within the notebook instead. In sum, Dan needs a way to perform data wrangling for his analysis from within his notebook, and generate readable and modifiable code to increase his effectiveness in performing a data analysis and his trust in the tools that are helping him do so.

Teaching workflows: Now that this analysis is complete Dan wants to teach this workflow to help other data scientists at his company perform data wrangling tasks, but how? Dan decides to create a tutorial video to explain his workflow and begins recording his thoughts while exploring his computational notebook. Dan cuts a lot of content he thinks is extraneous and would make his video too long and tedious for viewers to watch, like when he had to search online for more context about the data he was analyzing or code up a web scraping script to gather supplemental data to assist in his analysis. Dan also misses a lot of the smaller details and approaches he took when reflecting on this work, since he is just returning to his finished analysis after he has completed it. Dan realizes that edited or polished tutorials might not really capture all the ins and outs of being an effective professional data scientist, and wishes he could show more to his teammates since if any members of Dan's team watch his video, they might not be able to learn from his workflow as effectively without this less polished content.

Broadcasting workflows: Finally, Dan wants to share his knowledge with not only members of his team, but help mentor novice data scientists all over the world by sharing his expert data scientist workflows with them. Dan knows that one popular method for doing this is to become a content creator on streaming platforms like Twitch or YouTube. So, Dan searches how to setup a livestream to learn about the equipment required and sees that there is a substantial array of equipment options that can enhance the fidelity of his stream to his preference. However, navigating through this complexity causes frustration for Dan since he just wants to share with everyone his workflow and interact with other data scientists and viewers who might want to be data scientists, but realizes he not only has to be a data scientist, but also fulfill the role of an entire production team when building, maintaining, and running his stream. Dan needs a way to navigate through this complexity without putting extraneous stress on himself so he can focus on the content he wants to share instead of the entirety of the production.

1.3 Thesis

The research in this dissertation seeks to remove these barriers faced by data scientists and programmers during their daily workflows, which include the tedium of data wrangling, the difficulty of recalling functions or searching for solutions online, and the difficulty in verifying that these wrangling tools are doing what practitioners think they are doing. Secondly, I want to enable these expert practitioners to become expert content creators and to share these workflows online. Through livestreaming and creating educational media like video tutorials or interactive guides, I am interested in the benefits of this new form of media and the challenges felt by content creators when creating and sharing content with viewers or even co-workers in remote work situations. To do this, we need research to inform the future of remote collaboration and learning tools.

This leads to my thesis that: *Being able to see and interact with technical workflows behind the scenes can help novices grow into experts, but existing user-friendly computational systems may hinder growth by obscuring important kinds of complexity.*

1.4 Contributions

This dissertation details several contributions in the form of a formative study to define design principles for data wrangling tools, a novel data wrangling tool evaluated through quantitative and qualitative analyses with data scientists, an analysis of livestream videos to discover and define educational elements, design ideas for future tools in educational livestreaming, the creation of the design space of livestreaming setups, and challenges still needing to be addressed by future systems in supporting expert content creators. A summary of these contributions listed

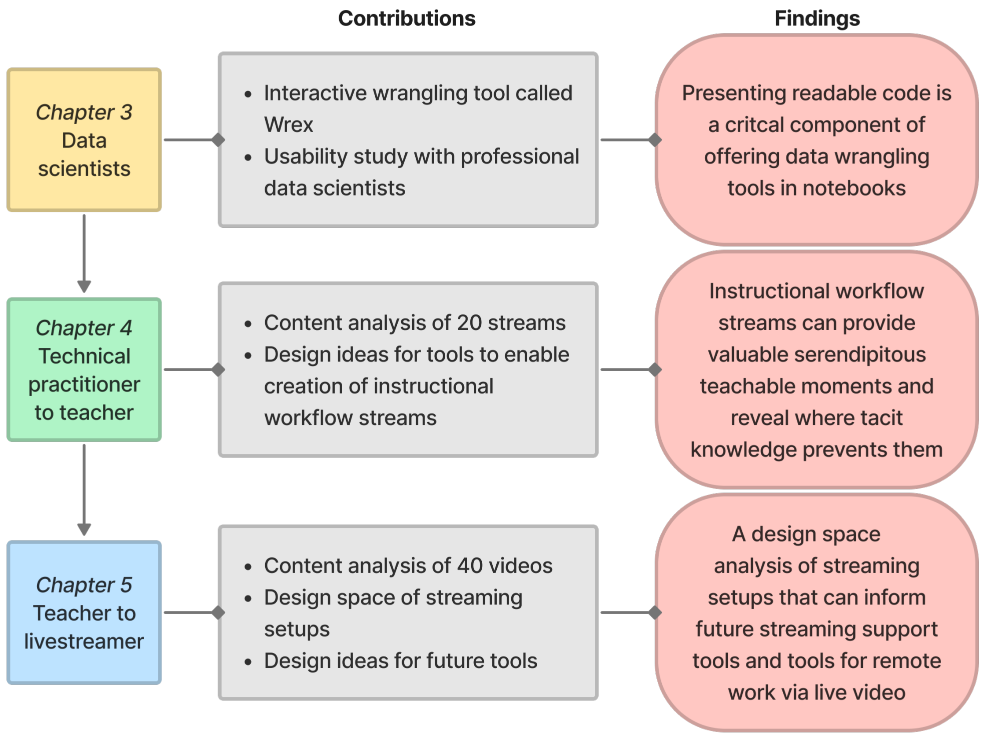

Contributions
Findings
Chapter 3
Data scientists
Interactive wrangling tool called Wrex
Usability study with professional data scientists
Presenting readable code is a critcal component of offering data wrangling tools in notebooks
Chapter 4
Technical practitioner to teacher
Content analysis of 20 streams
Design ideas for tools to enable creation of instructional workflow streams
Instructional workflow streams can provide valuable serendipitous teachable moments and reveal where tacit knowledge prevents them
Chapter 5
Teacher to livestreamer
Content analysis of 40 videos
Design space of streaming setups
Design ideas for future tools
A design space analysis of streaming setups that can inform future streaming support tools and tools for remote work via live video

Figure 1.4. Summary of the studies, contributions, and main findings in this dissertation

by chapter is below and in Figure 1.4:

In Chapter 3 we begin with a formative study with professional data scientists at Microsoft to determine design principles for future data wrangling systems. With these principles, we then propose Wrex, a hybrid interaction model that combines programming-by-example with readable code synthesis within the cell-based workflows in computational notebooks. Wrex provides wrangling-by-example through direct manipulation of an interactive dataframe, and is implemented as a Jupyter notebook extension to perform these user-determined data transformations. When a user interacts with these dataframes within their notebook, Wrex produces readable Python code that performs the transformations desired by the user so that users can inspect, modify, and use the synthesized code in their notebook or share and expand this code to other data analyses. The implementation of Wrex allows us to apply program synthesis to the domain of data science in a scalable way since Wrex allows data scientists to define their transforms on a snippet of their data and then apply this code to the larger, complete dataset (or even other datasets by incorporating this code into their existing data pipelines). We then perform a user study with professional data scientists at Microsoft to study the effectiveness of this interaction model. Our results showed that data scientists using Wrex were significantly more effective and efficient than when they manually programmed data wrangling tasks. Qualitative feedback provided insights in the effectiveness and acceptability of Wrex, including reducing barriers to data wrangling like searching for and recalling data transformation functions, verifying the tool is performing the task correctly by inspecting the readable Python code, and that Wrex fit right within their already established computational notebook data wrangling workflows.

In Chapter 4 we investigate the current use of livestreaming to teach cognitively complex skills like data science, programming, art, and gaming in a more naturalistic way. To enable this, we analyzed 20 videos we categorize as instructional workflow streams, or streams that show the longform workflows of these experts for the purpose of sharing their knowledge with novices. We propose the idea that this type of streaming can enable a virtual form of cognitive apprenticeship through working on real tasks and thinking aloud about their process. Our analysis found four different kinds of serendipitous teachable moments that lend themselves to cognitive apprenticeship, like improvised problem solving, improvised example creation, insightful tangents, and contextualized high-level advice. However, we also discovered several instances of missed teachable moments, possibly due to the inability to express tacit or unspoken expert knowledge. Finally, we propose the design of novel tools that can elicit this tacit knowledge to enable these streams to become more effective at cognitive apprenticeship and leveraging longer streams into more succinct and digestible tutorials.

In Chapter 5 we investigate the current state of livestreaming setups, which have a wide range of configurations based on budget and preferred production quality. We performed a survey of 40 videos published by streamers to describe various aspects of streaming setups, including the setups they themselves use, potential low- and high-fidelity setups, and the challenges involved in installing, running, and maintaining them as increasing fidelity in one area might require increasing the complexity of your setup in other areas. We then take our analysis of these videos to categorize current livestreaming equipment and create the design space of livestreaming setups which has ten dimensions that include: computers, software, stream control, encoding, cameras, lighting, video accessories, microphones, audio mixers, and audio accessories. We then discuss how this design space analysis can inform future streaming support tools to help streamers acquire, upgrade, and maintain new equipment for their own streaming setups. Finally, we explore how these findings might apply more broadly to improving live collaboration tools in a post-pandemic world where these tools for remote work and collaboration have become critical for companies throughout the world.

In Chapter 6, I conclude the dissertation with a summary of contributions held within. I then discuss the implications of these contributions, and the next steps for future work that these lines of research enable. Finally, I end with some final thoughts on balancing AI/ML systems with intelligent functionality without hiding the multitude of insights that users can learn from.

Chapter 2

Related Work

This dissertation builds on previous work in supporting data practitioners in performing data wrangling tasks through novel tools and program synthesis techniques, and is inspired by work in supporting teaching and sharing of complex workflows through the use of video tutorials and livestreaming. This chapter discusses related work and how this dissertation extends on the lessons learned from them.

2.1 Supporting data practitioners with their workflows

Chapter 3 introduces WREX, which extends and coalesces two lines of prior work: data wrangling tools and program synthesis for structured data.

2.1.1 Data Wrangling Tools

One well-known class of tools tackles the need to make data wrangling (e.g., preprocessing, cleaning, transformation [74]) more efficient. OpenRefine provides an interactive grid that allows the data scientist to perform simple text transformations, such as trimming a string, to clean up data columns, and to discover needed transformations through filters [44]. JetBrains' DataLore provides a data science IDE that provides code suggestions given user intentions [66]. Wrangler lowered the time and effort that data scientists spent on data wrangling by suggesting contextually relevant transforms to users, showing a preview window with the transform's effects, and providing an export to JavaScript function [73] (but this feature has since been removed in the commercial version [137]). Proactive Wrangler extended it with mixed-initiative suggested steps to transform data into relational formats [54].

Tempe provides interactive and continuous visualization support for live streaming data [27], where not only did the visualization change with new incoming data, but also changed with new user input through live programming support. TrendQuery is a "human-in-the-loop" interactive system that allowed users to iteratively and directly manipulate their data visualizations for the curation and discovery of trends [71]. Northstar describes an interactive system for data analysis aimed at allowing non-data-scientist domain experts and data scientists to collaborate, making data science more accessible [81]. DS.js leverages existing web pages, and the tables and visualizations on them, to create programming environments that help novices learn data science [153].

While these tools all provide increased efficiency for data scientists performing data wrangling tasks, they are missing several key features that Wrex provides. Most notably, they do not aim to generate readable code or to integrate with data scientists' existing workflows. Wrex uses program synthesis to achieve these goals and integrates with Jupyter, a popular computational notebook used by data scientists [78]. Through both our formative and controlled lab studies, we found these features to be critical for data scientists. Participants required saving source code as an artifact of their data wrangling so they could perform similar transformations on future datasets. Further, they wanted to take their wrangling scripts they created with their sample dataset and apply them to the full dataset in cluster or cloud environments.

As for notebook integration, Kandel et al. described a common data science workflow of context switching between raw data, wrangling tools, and visualization tools; Kandel noted that the "ideal" tool would combine these workflows into one tool [72]. We also found that data scientists desired tools such as Wrex that integrated with their current workflow. Using separate applications to perform data wrangling and analysis requires extra time and effort to import data into independent tools to wrangle their data, after which they will need to export their transformed data back into their preferred tool for data exploration, the computational notebook.

2.1.2 Program Synthesis for Structured Data

Gulwani et al. developed a new language and program synthesis algorithm implemented in Microsoft Excel that can perform several tasks that users have difficulty with in spreadsheet environments [48, 50, 51]. This feature, which became to be known as FlashFill, leveraged input-output examples defined by the user. FlashFill took these examples and created programs to perform string manipulations quickly, and with very few output examples from the user. Harris and Gulwani then applied this research direction to table transformations in spreadsheets [58]. Yessenov et al. used programming-by-example to do text processing [152]. Le and Gulwani then developed FlashExtract, a framework that uses examples to extract data from documents and tabular data [83]. Others have also leveraged synthesis to perform data transformations involving tabular data [38, 60, 67, 150].

This procession of research allows end-users to perform the above tasks without knowing how to write wrangling scripts. Further, even when users have the knowledge to create these scripts, these methods can produce results in a fraction of the time it takes to code these scripts by hand. This increases both the accessibility and efficiency of users dealing with data. WREX leverages these benefits to allow data scientists to forgo writing data wrangling scripts and focus on providing example output of desired data transformations.

These projects found examples to be “the most natural” way to provide a program synthesizer with a specification, but challenges remained in designing programming-by-example interaction models, particularly in user intent [49]. They noted that user examples may be ambiguous, so users need a way to address this ambiguity. WREX uses the “User Driven Interaction” model described in this line of work, which allows the user to examine the artifact, through reading the synthesized source code, and the behavior of the artifact, through the resultant output in the derived column. If any discrepancies exist with either, the user can provide further input by interacting with their data frame or by directly editing the code. Chasins et al. found that some participants perceived PBE (programming-by-example) tools to have less flexibility than traditional programming [12]. In Wrex, users have both the speed and ease-of-use benefits of PBE with the freedom to always switch to traditional text-based coding if the user perceives it to be necessary.

2.1.3 Data Science Challenges

However, beyond these two lines of research, some challenges still remain that inform Chapter 3 as researchers have investigated the challenges facing data science and scientists. Marx posits that software tools for data scientists need to be accessible to scientists without a formal software engineering background, produce common and compatible output, and be stable [103]. Wrex produces usable Python code that can be executed inside the data scientist's computational notebook or extracted to any Python script. In doing so, computer-novice scientists can bypass many of the roadblocks that occur during data wrangling. Further, data scientists can use the synthesized code to help them learn how to perform the required transformations.

Kandel et al. found that there are different proficiency levels of data scientists, leading to different tools and languages being used for analysis, and these data scientists can be categorized into several different archetypes of hackers, scripters, and application users [74]. We also saw evidence of an archetype system during our interviews with data scientists. Data scientists with a software engineering background generally preferred performance-focused and readable code so they could write up their own scripts, while data scientists with statistics or non-computational sciences backgrounds put less emphasis on performance and code readability. These data scientists were more concerned with the synthesized output effects being verifiable, but also found alternatives to readable code as an acceptable solution to verifiability.

2.1.4 Cognitive Load of Data Wrangling

Guo found that data scientists viewed data cleaning as a necessary, but "tedious and time-consuming" task [53]. Earlier reports have this tedious task taking up 50-80 percent of their time [89] [25]. Wickham posited that these data cleaning tasks reoccur frequently, and that they are critical in making data analysis easier [143]. Wrex lessens the burden on data scientists by assisting them in these data wrangling tasks by automating the writing of code that can perform the example transformations they provided, saving time and effort. This also removes the need to time-consuming context-switch between data science and programming, and the searching of solutions when recall of the necessary regex or Python function fails. This lowers the cognitive load of the data scientist by allowing them to remain focused on their data wrangling tasks.

Wickham also raises an issue with the ability to integrate analysis tools, as the output of one tool may not be compatible with another without human intervention [143]. Wrex's output has a high breadth of compatibility, as it generates Python source code and can directly manipulate the data frame passed by the data scientist.

Green posited that the way information is structured (and can be manipulated) enables different mental processes. The structure of these "notations" can make certain tasks easier or harder [46]. We believe that, through the direct manipulation of data frames and the generation of code to perform these manipulations, Wrex allows the data scientist to remain focused on the data wrangling tasks at hand and facilitates their completion.

Blackwell noted that programming-by-example systems should maintain visibility to the user by showing (1) the regex that is created by the system, (2) the input example that system is using to create a solution program, and (3) the effect of the regex on the data [5]. Wrex maintains these features as well as allows the user to modify the output code or regex and the input examples.

2.2 Supporting practitioners in teaching and sharing their workflows

These sections of related work relate to chapters 4 and 5 which include the analysis of livestreaming and screencast videos to determine their pedagogical use and livestreaming setup videos to form a design space of livestreaming setups.

2.2.1 Livestreaming motivation, challenges, and how we can learn from them

Over the past decade, livestreaming has emerged as a popular form of online media where content creators stream their naturalistic activities on platforms such as Twitch and YouTube. The most popular kinds of streamed activities include gaming, digital art, design, programming, physical crafts, music, calligraphy, and casual socializing [15, 37, 40, 91, 130, 141]. Screencasting (recording one's computer screen while working) is a related activity that is like livestreaming without an audience (a 'non-live' stream). Studies found that software developers upload screencasts to serve as video documentation or to build a public online following on platforms such as YouTube [98, 99]. HCI researchers have developed tools to facilitate live viewer interactions with streamers via multimedia-enhanced chat [13, 84, 151] and curated chat summaries [79, 93].

Streamers are motivated by goals such as entertainment, socializing, community-building, cultural heritage, financial interests, and teaching [40, 91, 93, 141]. For our study, we focus on the subset of streams that are purposely created with educational intent in mind: We call these *instructional workflow streams* in Chapter 4 since they come from a streamer demonstrating their naturalistic workflow with the intent of teaching technical skills to others. (Examples of *non*-instructional streams include someone playing a video game or a musical instrument for hours without any instructional commentary.)

Prior research on educational streams used surveys and interviews to discover the motivations of and challenges faced by streamers and viewers [2, 14, 15, 37, 40, 93]. We extend this emerging line of research by being the first, to our knowledge, to *perform an in-depth analysis of the pedagogical content within stream videos* in Chapter 4; our analysis is informed by the theoretical lens of cognitive apprenticeship [21, 124].

The closest related work are studies of programming streamers, which corroborate parts of our findings using complementary approaches (e.g., interviews and surveys). For instance, Chen et al. found that viewers enjoyed learning "over-the-shoulder" by hearing experts articulate their thought processes out loud and even seeing streamers make mistakes [15]. Alaboudi and LaToza [2] and Faas et al. [37] analyzed a sample of archived stream videos and reported instances of viewers providing debugging help in the live chat, along with general knowledge transfer of programming concepts. Haaranen reported similar kinds of Q&A happening in chats [56]. We also found these high-level trends in streams, but our study in Chapter 4 makes four novel contributions to this literature that extends prior work: 1) We propose cognitive apprenticeship [21, 22] as a theoretical lens through which to view instructional streams. 2) We discovered four types of *serendipitous teachable moments* that exemplify the improvised nature of streaming (Section 4.3), 3) we identified missed opportunities for additional knowledge transfer due to tacit knowledge (Section 4.4), 4) we covered four technical domains (web development, data science, digital art, gaming) rather than only programming.

HCI researchers have also studied the broader category of online tutorial videos, also sometimes called 'how-to videos' [77]. Unlike more conceptual lecture videos, tutorial videos aim to teach specific procedural skills via step-by-step demonstrations. They vary widely in format, ranging from physical task demonstrations [18, 77] (e.g., woodworking) to screencasts of software application usage [17, 47, 108] (e.g., Photoshop) to digitized handwritten sketches [24, 127] (e.g., Khan Academy math tutorials). HCI researchers have also developed tools that make it easier to create [17, 108], annotate [24, 108], and navigate [41, 76, 88, 106, 116] tutorial videos.

The instructional workflow streams we study in Chapter 4 are also meant to teach procedural 'how-to' skills. However, they embody the *opposite* of best-practices for tutorial videos: they are longform, unscripted, often unedited, and show an authentic end-to-end workflow rather than demonstrating on small pre-made examples. To our knowledge, no prior work has performed a media content analysis of instructional workflow streams. Our findings point toward tool design ideas that can help creators structure and repurpose their instructional workflow streams to gain some of the benefits of traditionally-produced tutorials but with lower production costs.

2.2.2 Studies of livestreamers and their viewers

Many researchers have also studied the activities of both streamers and their viewers. We found that these studies fall into four main clusters, each of which inspired the work in Chapter 5 in different ways:

1) Some focus on one specific domain that is popular amongst livestreamers, usually conducting surveys or interviews with both streamers and viewers in that domain. Popular domains of study include computer gaming [86, 101, 113, 126, 131], digital art [40], computer programming [2, 15, 36, 37], online education [14, 16, 28, 37, 56], and IRL (In Real Life – e.g., walking around) mobile streams [134] to showcase outdoors scenery [92] and preserve cultural heritage [91]. We were motivated by this prior work to include several popular streaming domains in our video content analysis, most notably gaming, digital art, and programming. However, mobile and IRL streams are out of scope for our study since we chose to focus on desktop equipment setups.

2) Some studies zoom in on the experiences of specific demographics of streamers, such as lifestyle streamers in China [96], those who use virtual 2D or 3D avatars instead of their faces (i.e., VTubers) [95], and streamers with visual impairments [68]. These mentioned aspects of streaming equipment within the context of streamers' overall experiences. For instance, Jun et al. reported on equipment challenges faced by streamers with visual impairments [68], such as interfacing between streaming software, screen readers, and text-to-speech apps, along with accessibility concerns with certain streaming software. Lu et al.'s study of VTubers found that widely-available software for controlling 2D avatar animations limited their expressiveness and that more professional streamers could afford full-body motion capture equipment that made their 3D avatars more expressive [95]. In contrast, our study zooms out to categorize more general-purpose streaming equipment such as computers, cameras, and microphones.

3) Researchers have also studied the motivations of livestream viewers [65, 130], how they support streamers both emotionally and financially [149], and emotional reactions to co-watching

streams together with a large audience [97]. Our study is inspired by some of their findings about the critical importance of streamers engaging with their audience live; what makes livestreaming unique is that it is more than simply recording pre-made videos. Thus, streamers need to adopt a set of equipment (constrained by their personal budget) that allows them to quickly switch scenes, interact with the chat, and show their real-time facial reactions to make viewers feel engaged.

4) Since livestreaming offers the possibility of streamer-viewer interactions, researchers have studied these dynamics of real-time performance [61, 85] as well as how monetary incentives (e.g., cash tips and digital gifts sent by viewers) affect streamer behavior [141]. Also, since viewers can interact with streamers via text chat, researchers have studied how volunteer moderators manage these chat channels to mitigate abuse and harassment [6, 10, 148]. Some studies mention streaming software extensions for moderating the chat, managing donations, and displaying on-screen notifications to enhance streamer-viewer interactions. These tools inspired us to add a *software/platform extensions* category to Section 5.3.1.

The study in Chapter 5 follows this lineage of livestreaming studies, but instead of focusing on streamers and their viewers like prior work has done, we turn our attention to the actual equipment that streamers use in practice. Prior work has mostly focused on motivations and challenges faced by streamers (and viewers) with regard to the actual topics they are streaming (although some touch upon specific aspects of equipment, as mentioned above). In contrast, our work instead focuses on streamers' relationships with the hardware and software equipment they use. To our knowledge, our study is the first to formulate a design space of the current state of desktop livestreaming equipment setups.

2.2.3 HCI systems to support livestreaming

Besides studying the current practices of streamers and viewers, HCI researchers have also built new interactive systems to enhance the livestreaming experience.

One major category of systems research involves enhancing the text chat communication channel between streamers and viewers. For instance, Chen et al. [13] augmented text chat with audio, video, and image stickers. Helpstone extends the chat features of the Hearthstone game to make it easier for stream viewers to give in-game hints and feedback [84]. Snapstream enables viewers to take, annotate, and post screenshots to the chat stream [151]. VisPoll allows a group of viewers to directly make visual inputs atop the video stream [19]. Similarly, StreamSketch lets streamers and viewers of art streams interact via a mix of sketching and text annotations [94].

Another category of systems aims to summarize the contents of streams to reduce information overload. For instance, Fraser et al. developed algorithms to split livestreams of people using creative tools (e.g., image editors) into meaningful segments that can be used to create shorter clips or tables of contents [39]. Kobs et al. fine-tuned sentiment analysis for stream text chats, which can help streamers feel more engaged with real-time audience reactions [79]. StreamWiki enables viewers to collaboratively summarize livestream contents for longer-term archiving. [93]. Finally, recent systems extend livestreaming beyond desktop and mobile settings. One representative example here is XRStudio, a platform for building lecture livestream experiences for instructors and students within virtual reality [109].

Our work in Chapter 5 differs from these systems projects because we aim to survey the *current state of practice of what livestreamers use in-the-wild.* These aforementioned papers describe novel research prototypes that have not been turned into widely-used products; thus, we did not include them into our design space analysis (Figure 5.2) since we did not observe streamers using them in the 40 videos we analyzed (Table 5.1). One area for future work is to create a combined design space with both research prototypes from academic papers and tools that are used by practitioners.

2.2.4 HCI survey papers that formulate design spaces

Methodologically, the study in Chapter 5 is most closely-related to HCI survey papers [147] that formulate a *design space* to capture different dimensions of variation in technical system features [82].

Traditional survey papers perform a systematic literature review to map out the landscape of academic work in a subfield. For instance, Brudy et al. reviewed 510 papers to formulate a comprehensive design space of multi-device interactions [8], with dimensions such as scale, temporal synchronicity, and user relationships. Frich et al. categorized 143 papers into a design space of creativity support tools [42], with dimensions such as device, phase of creativity process, and target user group. Pfeil et al. reviewed 52 papers to synthesize a design space of remote telepresence systems [114], with dimensions such as camera type, camera placement, and viewer communication.

Some HCI survey papers also analyze widely-used software tools, blog posts, product websites, and other popular media in order to capture systems used outside of academia. For instance, Lau et al. categorized 16 academic papers, 29 industry product websites, and 15 open-source software projects into a design space of computational notebook tools [82] with 12 dimensions, including data source, execution order, and versioning granularity. Similarly, Terrenghi et al. analyzed a few dozen academic papers and commercial products (e.g., touch-screen kiosk monitors) to formulate a design space of interactive display technology [135], with dimensions such as size, interaction modalities, and types of supported social interactions. Segel and Heer analyzed the content of 58 interactive visual storytelling webpages (e.g., New York Times interactive articles) to formulate a design space of narrative visualizations [125] consisting of 7 dimensions, including transition types, ordering, and highlighting. Striner et al. analyzed design process documents produced in a university gaming course to formulate a design space of audience participation needs for Twitch gaming livestreams [133], with dimensions such as agency, pacing, and community.

Our study in Chapter 5 follows in this tradition of practitioner-oriented design space analyses since we analyzed 40 streamer-made videos and the websites of products mentioned in those videos to formulate a design space of livestreaming equipment setups. We chose this approach (detailed in Section 5.2) since we wanted to map the state of current practice in the field rather than surveying academic papers.

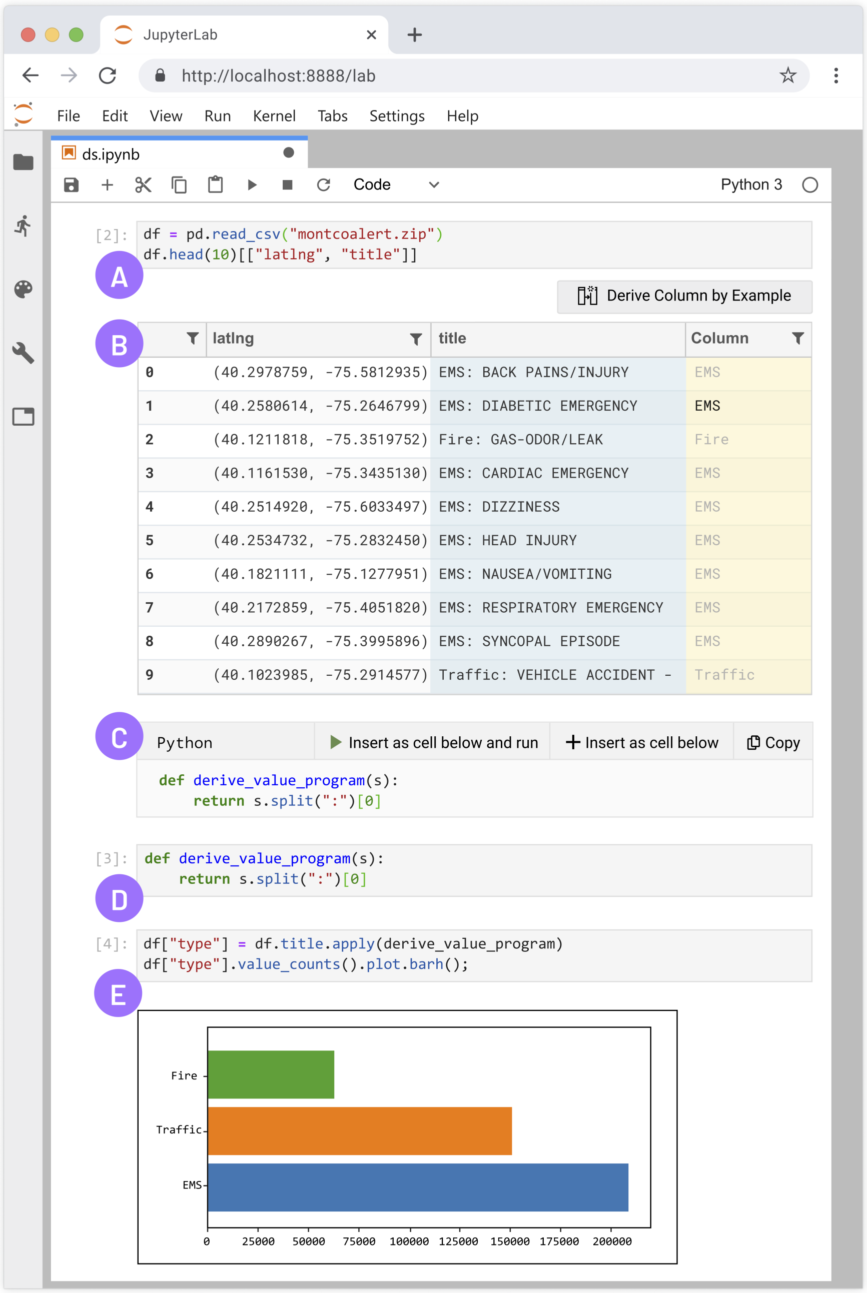
Chapter 3

Wrex: A Unified Programming by Example Interaction for Synthesizing Readable Code for Data Scientists

Data wrangling is a difficult and time-consuming activity in computational notebooks, and existing wrangling tools do not fit the exploratory workflow for data scientists in these environments. We propose a unified interaction model based on programming-by-example that generates readable code for a variety of useful data transformations, implemented as a Jupyter notebook extension called WREX*. User study results demonstrate that data scientists are significantly more effective and efficient at data wrangling with* WREX *over manual programming. Qualitative participant feedback indicates that* WREX *was useful and reduced barriers in having to recall or look up the usage of various data transform functions. The synthesized code allowed data scientists to verify the intended data transformation, increased their trust and confidence in* WREX*, and fit seamlessly within their cell-based notebook workflows. This work suggests that presenting readable code to professional data scientists is an indispensable component of offering data wrangling tools in notebooks.*

3.1 Introduction

Data wrangling—the process of transforming, munging, shaping, and cleaning data to make it suitable for downstream analysis—is a difficult and time-consuming activity [25, 54].



```
[2]: df = pd.read_csv("montcoalert.zip")
     df.head(10)[["latlng", "title"]]
```

	latlng	title	Column
0	(40.2978759, -75.5812935)	EMS: BACK PAINS/INJURY	EMS
1	(40.2580614, -75.2646799)	EMS: DIABETIC EMERGENCY	EMS
2	(40.1211818, -75.3519752)	Fire: GAS-ODOR/LEAK	Fire
3	(40.1161530, -75.3435130)	EMS: CARDIAC EMERGENCY	EMS
4	(40.2514920, -75.6033497)	EMS: DIZZINESS	EMS
5	(40.2534732, -75.2832450)	EMS: HEAD INJURY	EMS
6	(40.1821111, -75.1277951)	EMS: NAUSEA/VOMITING	EMS
7	(40.2172859, -75.4051820)	EMS: RESPIRATORY EMERGENCY	EMS
8	(40.2890267, -75.3995896)	EMS: SYNCOPAL EPISODE	EMS
9	(40.1023985, -75.2914577)	Traffic: VEHICLE ACCIDENT -	Traffic

Python

```
def derive_value_program(s):
    return s.split(":")[0]
```

```
[3]: def derive_value_program(s):
         return s.split(":")[0]
```

```
[4]: df["type"] = df.title.apply(derive_value_program)
     df["type"].value_counts().plot.barh();
```

Figure 3.1. WREX is a programming-by-example environment within a computational notebook, which supports a variety of program transformations to accelerate common data wrangling activities. Ⓐ Users create a data frame with their dataset and sample it. Ⓑ WREX's interactive grid where users can derive a new column and give data transformation examples. Ⓒ WREX's code window containing synthesized code generated from grid interactions. Ⓓ Synthesized code inserted into a new input cell. Ⓔ Applying synthesized code to full data frame and plotting results.

Consequently, data scientists spend a substantial portion of their time preparing data rather than performing data analysis tasks such as modeling and prediction.

Increasingly, data scientists orchestrate all of their data-oriented activities—including wrangling—within a single context: the computational notebook [3, 11, 26, 69, 75, 107, 110]. The notebook user interface, essentially, is an interactive session that contains a collection of input and output "cells." Data scientists use input code cells, for example, to write Python. The result of running an input cell renders an output cell, which can display rich media, such as audio, images, and plots. This interaction paradigm has made notebooks a popular choice for exploratory data analysis.

Through formative interviews with professional data scientists at a large, data-driven company, we identified an unaddressed gap between existing data wrangling tools and how data scientists prefer to work within their notebooks. First, although data scientists were aware of and appreciated the productivity benefits of existing data wrangling tools, having to leave their native notebook environment to perform wrangling limited the usefulness of these tools. Second, although we expected that data scientists would only want to complete their data wrangling tasks, our participants were reluctant to use data wrangling tools that transformed their data through "black boxes." Instead, they wanted to inspect the code that transformed their data. Crucially, data scientists preferred these scripts to be written in their familiar data science languages, like Python or R. This allows them to insert and execute this code directly into their notebooks, modify and extend the code if necessary, and keep the data transformation code alongside their other notebook code for reproducibility.

To address this gap, we introduce a hybrid interaction model that reconciles the productivity benefits of *interactivity* with the versatility of *programmability*. We implemented this interaction model as WREX, a Jupyter notebook extension for Python. WREX automatically displays an interactive grid when a code cell returns a tabular data structure, such as a data frame. Using programming-by-example, data scientists can provide examples to the system on the data transform they intend to perform. From these examples, WREX generates *readable* code in Python, a popular data science language.

Existing programming-by-example systems for data wrangling address some, but not all, of these requirements. FlashFill [58] does not display the transformed code to the data scientist. Although Wrangler [54, 73] can produce Python code, these scripts are not designed to be read or modified directly by data scientists. Trifacta [137] produces readable code, but in a domain-specific language and not a general-purpose one. The contributions of this chapter are as follows:

- We propose a hybrid interaction model that combines programming-by-example with readable code synthesis within the cell-based workflow of computational notebooks. We implement this interaction model as a Jupyter notebook extension for Python, using an interactive grid and provisional code cell.

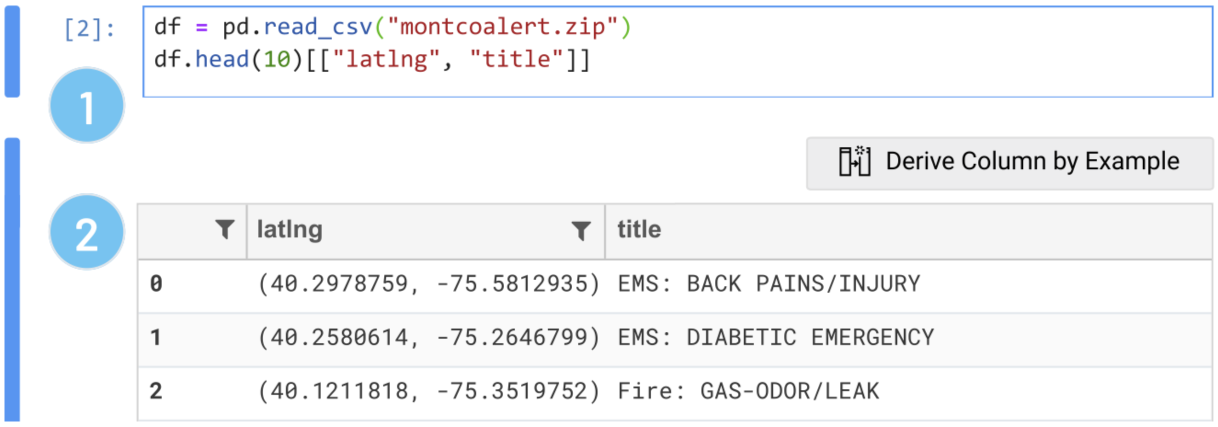
- We apply program synthesis to the domain of data science in a scalable way. Up until now, program synthesis has been restricted to Excel-like settings where the user wants to transform a small amount of data. Our approach allows data scientists to synthesize code on subsets of their data and to apply this code to other, larger datasets. The synthesized code can be incorporated into existing data pipelines.

- Through a user study, we find that data scientists are significantly more effective and efficient at performing data wrangling tasks than manual programming. Through qualitative feedback, participants report that Wrex reduced barriers in having to recall or look up the usage of various data transform functions. Data scientists indicated that the availability of readable code allowed them to verify that the data transform would do what they intended and increased their trust and confidence in the wrangling tool. Moreover, inserting synthesized code as cells is useful and fits naturally with the way data scientists already work in notebooks.

3.2 Example usage scenario for Wrex

Dan is a professional data scientist who uses computational notebooks in Python. He has recently installed Wrex as an extension within his notebook environment. Dan has an open-ended task that requires him to explore an unfamiliar dataset relating to emergency calls (911) for Montgomery County, PA. The dataset contains several columns, including the emergency call's location as a latitude and longitude pair, the time of the incident, the title of the emergency, and an assortment of other columns.

First Steps: As with most of his data explorations, Dan starts with a blank Python notebook. He loads the `montcoalert.zip` dataset into a data frame using pandas—a common library for working with this rectangular data. He previews a slice of the data frame, the `latlng` and `title` columns for the first ten rows (A) (1). Wrex displays an interactive grid representing the returned data frame (B) (2). Through the interactive grid, Dan can view, filter, or search his data. He can also perform data wrangling using "Derive Column by Example."

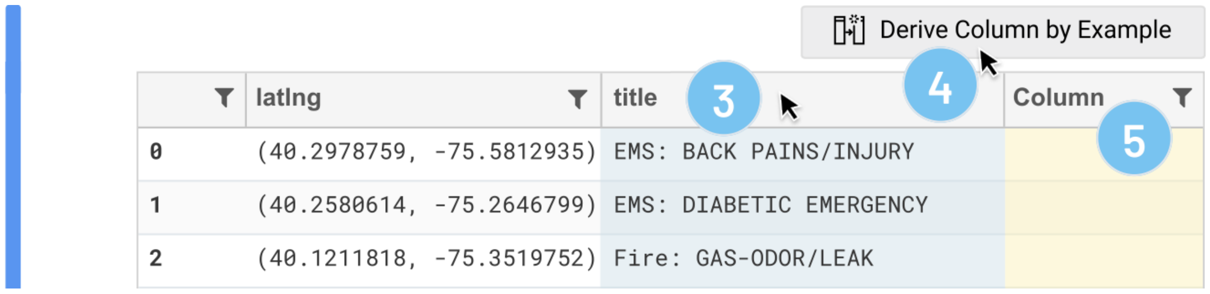


```
df = pd.read_csv("montcoalert.zip")
df.head(10)[["latlng", "title"]]
```

	latlng	title
0	(40.2978759, -75.5812935)	EMS: BACK PAINS/INJURY
1	(40.2580614, -75.2646799)	EMS: DIABETIC EMERGENCY
2	(40.1211818, -75.3519752)	Fire: GAS-ODOR/LEAK

From Examples to Code: Dan notices that cells in the `title` column seem to start with `EMS`, `Fire`, and `Traffic`. As a sanity check, he wants to confirm that these are the only types of incidents in his data, and also get a sense of how frequently these types of incidents are happening.

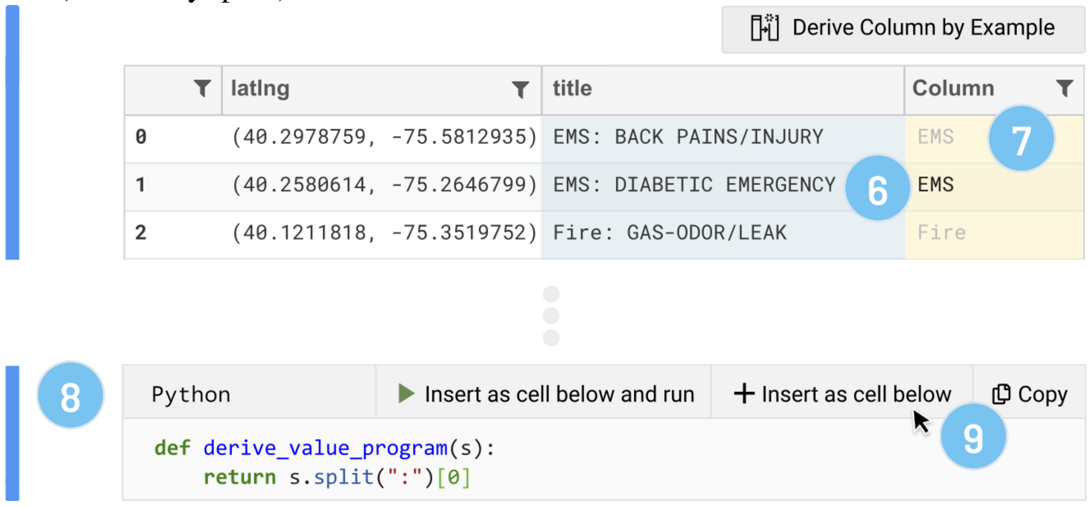
Dan selects the `title` column by clicking its header (3), then clicks the "Derive column by example" button to activate this feature (4). The result is a new empty column (5) through which Dan can provide an example (or more, if necessary) of the transform he needs.



	latlng	title	Column
0	(40.2978759, -75.5812935)	EMS: BACK PAINS/INJURY	
1	(40.2580614, -75.2646799)	EMS: DIABETIC EMERGENCY	
2	(40.1211818, -75.3519752)	Fire: GAS-ODOR/LEAK	

He arbitrarily types in his intention, EMS, into the second row of the newly created column in the grid 6. When Dan presses Enter or leaves the cell, WREX detects a cell change in the derived column. WREX uses the example provided by Dan (EMS) with the input example taken from the derived from column `title` (EMS: DIABETIC EMERGENCY) to automatically fill in the remaining rows 7.

In addition, WREX presents the actual data transformation to Dan as Python code through a *provisional code cell* C 8. This allows Dan to inspect the code Python code before committing to the code. In this case, the code seems like what he probably would have written had he done this transformation manually: split the string on a colon, and then return the first split. Dan decides to insert this code into a cell below this one, but defers executing it 9. If Dan had actually intended to uppercase all of the types, he could have provided WREX with a second example: `Fire` to `FIRE`. If desired, Dan could have also changed the target from Python to R for comparison (or even PySpark), since Dan is a bit more familiar with R.



	latlng	title	Column
0	(40.2978759, -75.5812935)	EMS: BACK PAINS/INJURY	EMS
1	(40.2580614, -75.2646799)	EMS: DIABETIC EMERGENCY	EMS
2	(40.1211818, -75.3519752)	Fire: GAS-ODOR/LEAK	Fire

```python
def derive_value_program(s):
    return s.split(":")[0]
```

Since the new input cell is just Python code Ⓓ, Dan is free to use it however he wants: he can use it as is, modify the function, or even copy the snippet elsewhere. Dan decides to apply the synthesized function to the larger data frame—this results in adding a `type` column to the data frame (`df`). He plots a bar chart of the count of the categories and confirms that there are only three types of incidents in the data Ⓔ.

From Code to Insights: Having wrangled the `title` column to `type`, and given the `latlng` column already present, Dan thinks it might be interesting to plot the locations on a map. To do so, the `latlng` column is a string and needs to be separated into `lat` and `lng` columns. Once again, he repeats the data wrangling steps as before: Dan returns a subset of the data, and uses the interactive grid in WREX to wrangle the latitude and longitude transforms out of the `latlng` column. He applies these functions to his data frame. Having done the tricky part of data wrangling the three columns—`lat`, `lng`, and `type`—he cobbles together some code to add this information onto `folium` ⑩, a map visualization tool.

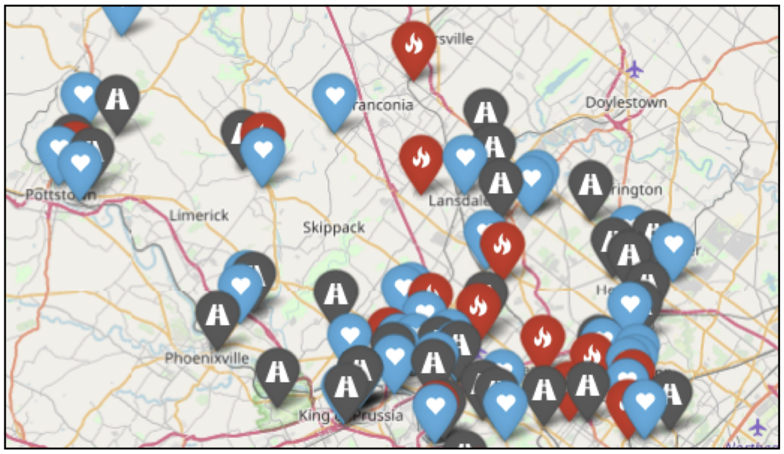
[9]:

⑩

```
import folium
m = folium.Map()

def plot(m, r):
    p = [r["lat"], r["lng"]]
    icon = folium_icon_lookup(r["type"])
    folium.Marker(p, icon=icon).add_to(m)

df[["lat", "lng", "type"]].apply(lambda r: plot(m, r), axis=1)
m
```



Like the data scientists in our study, Dan finds that data wrangling is a roadblock to doing more impactful data analysis. With WREX, Dan can accelerate the tedious process of data wrangling to focus on more interesting data explorations—all in Python, and without having to leave his notebook.

3.2.1 Working without WREX

Imagine instead if Dan had to use a Jupyter notebook without WREX. Dan first prints his data frame to observe the data. The data frame is a static table that Dan cannot manipulate or filter in any way. Dan inserts a new input cell to begin writing code to extract the desired strings from the "title" column. Dan does not know the Python code required do this so he begins a web search to find API and Stack Overflow pages that might be useful in helping him accomplish his goal. After spending some time looking for solutions he tries to adopt a code snippet into his notebook and modifies it to suit his needs, but Dan runs into syntax errors when trying to run the code. After making changes and debugging errors, he applies his code to his dataset and re-prints it. The output column he receives looks correct, but since the dataset has over a million entries it is not feasible to manually check every row for correctness. So, Dan moves on to visualizing his new column via a histogram and sees some of his categories are extracted incorrectly. Dan now has to inspect the offending rows and try to debug his code, taking more time away from his actual goal of data analysis.

Imagine instead if Dan had used a standalone GUI tool outside of his notebook. For example, Dan could load the dataset into Trifacta's Wrangler [137] to handle his wrangling tasks. Though Wrangler affords many useful features to enable data wrangling, Dan runs into several issues. Dan must leave his notebook environment he prefers to do data analysis in, and do data wrangling in the Wrangler environment. After Dan completes his data wrangling, he must first export the changed dataset, and then reload the dataset back into his notebook. In order to do his data wrangling, Dan must learn to use a new domain-specific language (DSL) unique to Wrangler, rather than stay in his preferred language of Python. Beyond this, using a unique

DSL affects collaboration with the other data scientists that Dan works with. The script required to clean the dataset exists within Wrangler, meaning all of Dan's collaborators must also use Wrangler to follow Dan's workflow, rather than remain in their preferred notebook environments, using their preferred programming language, and using their preferred data science libraries.

3.3 Formative interviews and design goals

We conducted interviews with seven data scientists who frequently use computational notebooks at a large, data-driven software company. In our interviews, we focused on how they perform data wrangling, how data wrangling fits within their notebook workflow, what tools they use or have used for data wrangling, and what difficulties they face as they wrangle data. These data scientists (F1–F7) provided several insights that guided the design goals for WREX.

Data scientists reported that using standalone tools designed for data wrangling required "excessive roundtrips" (F2, F4) or "shuffling data back and forth" (F1, F6) between their notebooks and the data wrangling tool. As a result, they preferred to write their wrangling code by hand in their notebooks. F2 explained that although these tools have nice capabilities, "[they] shouldn't have to go somewhere else just to transform data." F6 wondered if was "possible to put some of these capabilities [that are available in standalone tools] within their notebooks." This feedback led to our first design goal:

> **D1.** Data wrangling tools should be available where the data scientist works—within their notebooks.

All of our participants wanted tools that produced code as an inspectable artifact, because, "as a black box; you don't have a good intuition about what is happening to your data" (F7) and because "black boxes aren't transparent, the data transforms aren't customizable. If the tool doesn't have your transformation, you have to write it yourself anyway" (F6).

Although some tools allowed data scientists to view their data transformations as scripts, we found that data scientists preferred that these scripts be written in languages they already were comfortable with (F1, F6). For example, F1 "preferred general-purpose languages for doing data science." F6 explained, "there's a learning curve to having to learn new libraries". F7 added that the scripts from these tools were often quite limited: "I'm an expert in Python; these [languages for data wrangling] seem to cater only to novice programmers. They don't compose well with our existing notebook code or the 'crazy formats' we have to deal with."

Data scientists' desire for inspectable code as output of the data transformation tool, their preference for using familiar programming languages, and the desire to customize or extend data wrangling transforms led to our second design goal:

> **D2.** Data wrangling tools should produce code as an inspectable and modifiable artifact, using programming languages already familiar to the data scientist.

3.4 Wrex system design and implementation

Wrex is implemented as a Jupyter notebook extension. The front-end display component is based on Qgrid [117], an interactive grid view for editing data frames. Several changes were made to this component to support code generation. First, we modified Qgrid to render views of the underlying data frame, rather than the data frame itself. Second, we added the ability to add new columns to the grid. By implementing both of these changes, users are able to give examples through virtual columns without affecting the underlying data. Third, we added a view component to Qgrid to render the code block. Finally, we bound to appropriate event handlers to invoke our program synthesis engine on cell changes. To automatically display the interactive grid for data frames, the back-end component injects configuration options to the Python pandas library [112] and overrides its HTML display mechanism.

3.4.1 Readable Synthesis Algorithm

The program synthesis engine that powers Wrex substantially extends the FlashFill toolkit [105], which provide several domain specific languages (DSLs) with operators that support string transformations [48], number transformations [129], date transformations [129], and table lookup transformations [128]. A technical report by Gulwani et al. [52] formally

Table 3.1. Wrex synthesizes readable code for transformations commonly used by data scientists during data wrangling activities. After selecting one or more columns (text in blue), the data scientists can specify examples in an output column to provide their intent (text in red). As the data scientist provides examples, Wrex generates a synthesized code block and presents this code block to them.

Transform	Input(s) / Example(s)	Synthesized Code
String		
Extracting	`12;L MERION;CITY AVE` `L MERION`	`a = s.index(";") + len(";")` `b = s.rindex(";")` `return s[a:b]`
Case manipulation	`NEW HANOVER` `New Hanover`	`return s.title()`
Concatenation	Claudio A Chew Claudio-A-Chew	`return "{}-{}-{}".format(s, t, u)`
Generating initials	`Doug` `Funnie` `D.F.`	`t = regex.search(r"\p{Lu}+", s).group(0)` `u = list(regex.finditer(` `r"\p{Lu}+", s))[-1].group(0)` `return "{}.{}.".format(t, u)`
Mapping const values	`Male` `0` `Female` `1`	`{ "Male": 0, "Female": 1 }.get(s)`
Number		
Round to two decimals with ties going away from zero	`-15.319` `-15.32` `17.315` `17.32`	`return Decimal(s).quantize(` `Decimal(".01"),` `rounding = ROUND_HALF_UP)`
Round up to nearest 100	`6512` `6600` `23` `100`	`return 100 * math.ceil(x / 100)`
Scaling	`-12.5` `-12500`	`return x * 1000`
Padding	`828` `00828`	`return str(n).zfill(5)`
Date and Time		
Extracting parts	`31-Jan-2031 05:54:18` `Fri`	`dt = strptime(s, "%d-%b-%Y %H:%M:%S")` `return dt.strftime("%a")`
Formatting	`2015-12-10 17:10:52` `10 Dec 2015`	`dt = strptime(s, "%Y-%m-%d %H:%M:%S")` `return dt.strftime("%d %b %Y")`
Binning	`2:02` `02:00-04:00`	`dt = datetime.strptime(s, "%m/%d/%Y %H:%M")` `base_value = timedelta(hours = dt.hour, ...)` `delta_value = timedelta(hours = 2)` `dt_str = (dt - base_value % delta_value) \` `.strftime("%H:%M")` `rounded_up_next = (dt - base_value % delta_value) \` `+ delta_value` `return dt_str + "-" + rounded_up_next.strftime("%H:%M")`
Composite		
Point Compose	`40.865324` `-73.935237` `(40.87, -73.94)`	`d1 = Decimal(s).quantize(` `Decimal(".01"), rounding = ROUND_HALF_UP)` `d2 = Decimal(t).quantize(` `Decimal(".01"), rounding = ROUND_HALF_UP)` `return "({}, {})".format(d1, d2)`
Fixed-Width Compose	`3` `71` `03071`	`s = str(n).zfill(2))` `t = str(n).zfill(3))` `return s + t`

describes the semantics of extensions; WREX uses these extensions, which we summarize in this section.

With WREX, we surfaced this PBE algorithm through an interaction that is accessible to data scientists. The algorithm supports a variety of transformations, and even compositions of those transformations, without requiring the user to explicitly specify any input or output data types. Table 3.1 lists examples of the resulting synthesized Python code for typical data science use cases; the synthesized code for EXTRACTING is only three lines of code. In the classic FlashFill algorithm, this same program is over 30 lines of code.

The extended FlashFill algorithm (RCS) has four phases:

Phase 1: Standard Program. RCS calls SYNTHESIZE with the user-provided examples, using the standard FlashFill ranker. Since the FlashFill ranker is optimized to minimize the number of required examples, data scientists can in many cases obtain a useful program (P_1) by giving only a single example. Here, the program is represented as an internal DSL.

Phase 2: Readable Program. We use the size of the program as a proxy for readability, and design a ranker that prefers small programs, which are likely easier to understand. This ranker is also designed to prefer programs that use DSL operators that have direct translation into the target language (e.g., Python). Since the readability ranker is optimized to prefer small programs, the ranker requires more examples than the FlashFill ranker. The insight is to apply the program P_1 to all required input columns in the data frame to obtain these additional examples (`examples_all`). RCS again calls SYNTHESIS to obtain the program P_2, this time using `examples_all` and the readability ranker. Concretely, consider the transform of “21-07-2012” to “21”. FlashFill (intent-based) takes the sub-string that matches `\d+` on the left and “-” on the right—because it handles dates like “4-12-2018” (`input.match('^\d+')`). However, tuning towards generality makes it less succinct. The objective-based ranker chooses `input[0:2]`, but if and only if the behavior matches on a much larger sample of inputs (maintaining behavioral equivalence to the intent-based ranker). Hence, we pick `input[0:2]` if there are no inputs of the form “4-12-2018”. If there are inputs in such a form, the user would have to provide a second

Algorithm 1. Program synthesis phases for readable code.

```
function ReadableCodeSynthesis(df, examples)
    $P_1 \leftarrow$ Synthesize(examples, flashfill_ranker)
    examples_all $\leftarrow \{(\text{row}, P_1(\text{row})) \mid \text{row} \in \text{df}\}$
    $P_2 \leftarrow$ Synthesize(examples_all, readability_ranker)
    $P_3 \leftarrow$ Rewrite($P_2$, rules, df)
    code $\leftarrow$ translate_to_target($P_3$)
    return Formatter(code)
```

example.

Phase 3: Rewriting. The goal of the rewriting phase is to transform the synthesized program into another program that is simpler to understand. As before, we apply the insight that we can use rewrite rules such that the synthesized program preserves the behavior of `examples_all`, but allows for changing the semantics of any potential inputs that have *not* been passed to RCS. One such rule rewrites "`[0-9]+(\,[0-9]{3})*(\.[0-9]+)?`" to "`\d+`". This replaces a complex pattern that matches numbers with commas and decimal point with a pattern that matches a sequence of digits. Clearly, replacing the first regular expression by second one will change the semantics of a program. But if all numbers in all the inputs are of the form "`\d+`", then the replacement will preserve behavioral equivalence.

Phase 4: Translation to the Target Language. The final translation step goes down the abstract syntax tree (AST) of the DSL-program, and translates each node (DSL operator) into the equivalent operator in the target language—which today can be Python, R, and PySpark. For example, the Concat operator in the DSL is just mapped to `+` or a `format` method on a string, depending on the number of elements concatenated. If the DSL operator does not have a semantically-equivalent Python operator, then the translator generates multiple lines of code in the target language to emulate its behavior. Finally, the target code is passed through an off-the-shell code formatter: for Python, this is `autopep8` [59].

3.4.2 Limitations

A limitation of Wrex is that user-friendly error handling is not implemented yet. Errors can arise in two ways: when the user specifies a conflicting set of constraints (for example, transforming “`Traffic` to `T`” alongside “`Traffic` to `TR`”), or when the synthesis engine fails to learn a program. Program synthesis will also unrelentingly generate incomprehensible programs due to difficult-to-spot typos in user-entered examples (such as having a trailing space in an example). In such cases, we asked participants to invoke the grid again and redo the task, although we did not restart their task time. In practice, it is unrealistic to expect that data scientists can perfectly provide examples to the system, so these issues will need to be addressed in future work. When the user introduces these internally conflicting examples or when rows in a dataset have ambiguous values (e.g., `null`), it is useful to suggest additional rows to investigate; this significant inputs feature is available but not evaluated in this chapter.

Some tasks are not amenable to programming and thus are not performed by Wrex, like certain natural language transformations (e.g., “S.F.” to “San Francisco”), and other tasks that require aggregation like the sum or average of the entire column. Another limitation comes from how a user samples their data (for example, `df.head(n)`, which may lack sufficient diversity in range of exposed values). This issue may lead to synthesized code that works perfectly for the sample but runs into issues on the full dataset.

Users may not know when to stop providing examples (where further examples have little effect on synthesis). Here users must inspect the data frame and code to determine if Wrex narrows in on an acceptable solution. It outputs only the top-ranked one, and it is possible that the user may prefer a lower-ranked program (e.g., uses non-regex instead of regex). Finally, Wrex is aimed for professional data scientists who work mostly within notebooks; users of Excel, Tableau, and other GUI tools may be more accustomed to switching between multiple tools, so an integrated single-app workflow may not be as necessary for them.

3.5 Evaluation: In-lab comparative user study

Participants: We recruited 12 data scientists (10 male), randomly selected from a directory of computational notebook users with Python familiarity within a large, data-driven software company. They self-reported an average of 4 years of data science experience within the company. They self-rated familiarity with Jupyter notebooks with a median of "Extremely familiar (5)," using a 5-point Likert scale from "Not familiar at all (1)" to "Extremely familiar (5)," and their familiarity with Python at a median of "Moderately familiar (4)."

Tasks: Participants completed six tasks using two different datasets. These tasks involved transformations commonly done by data scientists during data wrangling, such as extracting part of a string and changing its case, formatting dates, time-binning, and rounding floating-point numbers.

The first dataset, called A, contains emergency call data containing columns with dates, times, latitude, longitude, physical location with zip code and cross streets, and an incident description.[1] We designed three tasks using this dataset:

- A1 Using the `Location` (`19044;HORSHAM;CEDAR AVE & COTTAGE AVE`) column, extract the city name and title case it (`Horsham`).
- A2 Using the `Date` (`12/11/2015`) and `Time` (`13:34:52`) columns, format the date to the day of the week, time to 12-hour clock format, and combine these values with an "@" symbol (`Friday @ 1pm`).
- A3 Using the `Latitude` (`40.185840`) and `Longitude` (`-75.125512`) columns, round half up the values to the nearest hundredths place and combine them in a new format (`[40.19, -75.13]`).

The second dataset, called B, contains New York City noise complaint data which includes columns containing the date-timestamp of the call, the date-timestamp of when the

[1]https://www.kaggle.com/mchirico/montcoalert

incident was closed, type of location, zip code of incident, city of incident location, borough of incident location, latitude, and longitude.[2] We designed three tasks using this dataset:

B1 Using the `Created Date` (`12/31/2015 0:01`) column, extract the time and place it in a 2-hour time bin (`00:00-02:00`).

B2 Using the `Location Type` (`Store/Commercial`), `City` (`NEW YORK`), and `Borough` (`MANHATTAN`) columns, title case the values and combine them in a new format (`Store/Commercial in New York, Manhattan`).

B3 Using the `Latitude` (`40.865324`) and `Longitude` (`-73.938237`) columns, round half down the values to the nearest hundredths place and combine them in a new format (`(40.86, -73.94)`).

Protocol: Participants were assigned A and B datasets through a counterbalanced design, such that half the participants received the A dataset first (A-dataset group), and the other half received the B dataset first (B-dataset group). We randomized task order within each dataset to mitigate learning effects. They first completed three tasks with a Jupyter notebook (manual condition). They had 5 minutes per task to read the requirements of the task and write code to complete the task. Participants were provided a verification code snippet within their notebooks that participants ran to determine if they had completed the task successfully. If participants failed to complete the task within the allotted time, we marked the task as incorrect. Participants had access to the internet to assist them in completing the task if needed. At the end of the manual condition, we interviewed the participants about their experience and asked them to complete a questionnaire to rate aspects of their experience. Next, participants completed a short tutorial that introduced them to Wrex. After participants completed the tutorial, they moved on to the second set of tasks, this time using Wrex with conditions similar to the first set of tasks. After the three tasks are completed, we again interviewed them about their experience and asked them to complete the questionnaire.

[2]https://www.kaggle.com/somesnm/partynyc

Questionnaires: After the first set of tasks, participants rated how often the tasks showed up in their day-to-day work using a 5-point Likert scale from “Never (1)” to “A great deal (5)”, and discussed what aspects of the notebook made it difficult to complete the tasks and what affordances could address these difficulties. After the second set of tasks, this time with Wrex, the participants took a second questionnaire that also had them rate task representativeness, and asked free-form questions on difficulties they had and tool improvements. Further, the second questionnaire asked the participant to rate grid and code acceptability using a 5-point Likert-type item scale ranging from “Unacceptable (1)” to “Acceptable (5)”, and rate the likeliness they would use a productionized version of Wrex using a 5-point Likert-type item scale ranging from “Extremely unlikely (1)” to “Extremely likely (5)”. Finally, participants were interviewed after each set of tasks about their experience with Jupyter notebooks and Wrex.

Follow-up: We directly addressed participant feedback to improve the synthesized code: We removed the use of classes entirely and replaced these instances with lightweight functions. We replaced the register-based variable naming scheme (`_0` and `_1`) with a variable-name generation scheme that uses simpler mnemonic names, such as `s` and `t` for string arguments. We removed exception handling logic because these constructs made it harder for the data scientists to identify the core part of the transformation. Finally, we returned to the participants after implementing these changes and asked them to reassess the synthesized code for the study tasks.

3.6 Quantitative results

Table 3.2 shows completion rates by task and condition. Fisher’s exact test identified a significant difference between the Wrex and manual conditions, both in the A-dataset first ($p < .0001$) and in B-dataset first ($p < .0001$) subgroups. Participants in the manual condition completed 12/36 tasks, while those in the Wrex condition completed all 36/36 tasks. The significant differences between the two conditions can be explained mostly by tasks A2 and B1, which require non-trivial date and time transformations.

Table 3.2. Participant task completion under Wrex and manual data wrangling conditions. Participant reported frequency of tasks in day-to-day work. Participants were given five minutes to complete each task. Rating scale for task frequency from left-to-right: ▢ Never (1), ▢ Rarely (2), 🟦 Occasionally (3), 🟦 Moderately (4), 🟦 A great deal (5). Median values precede each distribution.

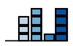
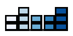
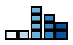
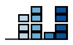
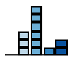
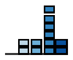
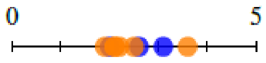
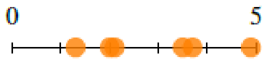
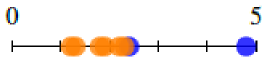
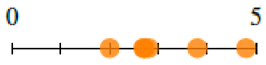
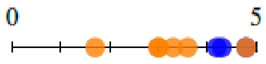
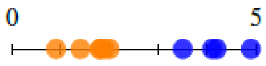
	Manual		Wrex		Frequency		
Task	*n*	%	*n*	%	*n*	Dist.	
A1	3	50%	6	100%	12	3	
A2	0	0%	6	100%	12	2	
A3	2	33%	6	100%	12	2	
B1	0	0%	6	100%	12	3	
B2	3	50%	6	100%	12	2	
B3	4	67%	6	100%	12	2	

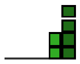
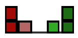
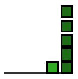
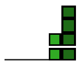
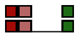
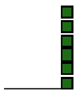
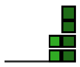
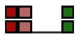
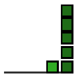
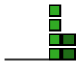
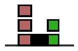
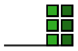
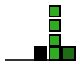
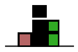
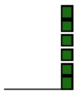
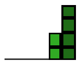
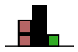
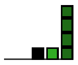
Table 3.3. Participant efficiency under ● Wrex and ● manual data wrangling conditions.

Task	**Timeline**	**Manual**		**Wrex**	
		n	Time (min)	*n*	Time (min)
A1		3	2.5	6	2.4
A2		0	5.0	6	2.9
A3		2	3.6	6	1.8
B1		0	5.0	6	3.1
B2		3	4.4	6	3.2
B3		4	4.2	6	1.7

Participant Efficiency: Table 3.3 shows the distribution of completion times by task and condition, and the participants' self-reported frequency of how often they do that type of task. A t-test failed to identify a significant difference in the A-dataset ($t(5.93) = 1.13$, $p = 0.30$), but did identify a significant difference for the B-dataset ($t(22.32) = 5.17$, $p < 0.001$). Using Wrex, the average time to completion was $u = 2.4$, $sd = 1.0$ (A) and $u = 2.7$, $sd = 1.0$ (B). Participants using Wrex, on average, were about 40 seconds faster ($u = 0.60$, $sd = 0.53$) in A, and about 1.6 minutes faster ($u = 1.61$, $sd = 0.31$) in B. This means if one has a good understanding of the code required to perform their transformation—and if the code is simple to write—then it may be faster to write code directly than to give an example to Wrex.

Grid and Code Acceptability: Table 3.4 shows distribution of acceptability for the grid, the code acceptability during the study (Code_1), and the code acceptability after post-study improvements (Code_2). Participants reported the median acceptability of the grid experience as Acceptable (5). The code acceptability during the study (Code_1) had substantial variation in responses, with a median of Neutral (3). After improving the program synthesis engine based on

Table 3.4. How acceptable was the grid experience and the corresponding synthesized code snippet? Rating scale from left-to-right: 🟥 Unacceptable (1), 🟫 Slightly unacceptable (2), ⬛ Neutral (3), 🟩 Slightly acceptable (4), and 🟩 Acceptable (5). Code$_1$ are the ratings from the code synthesized in the in-lab study. Code$_2$ are the ratings after incorporating the participants' feedback. Median values precede each distribution.

Task	Acceptability			
	n	Grid	Code$_1$	Code$_2$
A1	6	5 	3 	5 
A2	6	5 	2 	5 
A3	6	5 	2 	5 
B1	6	4 	2 	4 
B2	6	4 	3 	5 
B3	6	5 	3 	5 

the participant feedback (Code$_2$), the median score improved to Acceptable (5). A Wilcoxon signed-rank test identified these differences as significant ($S = 319, p < .0001$), with a median rating increase of 2. As a measure of user satisfaction, we asked participants if they would use WREX for data wrangling tasks if a production version of the tool was made available: five participants reported that they would probably use the tool (4), and seven reported that they would definitely use the tool (5).

3.7 Qualitative feedback: from study participants

3.7.1 Reducing Barriers to Data Wrangling

After completing the three tasks in the manual Jupyter condition, participants noted these sets of barriers to wrangling that they experienced both during the tasks and also in their daily work, some of which Wrex helped overcome.

Recall of Functions and Syntax: The most common barrier reported by participants, both within our lab study and in their daily work, is remembering what functions and syntax are required to perform the necessary data transformations. One reason for failed recall is due to lack of recency, "my biggest difficulty was recalling the specific command names and syntax, just because I didn't use them today" (P2). The complexity of modern languages and the number of libraries available is too vast for data scientists to rely on their memory faculties as "it is just tough to memorize all the nuances of a language" (P5). Participants noted that although computational notebooks have features like inline documentation and autocompletion, these features don't directly help them in understanding which operations they need to use and how they should use them.

Wrex reduces this barrier with the synthesis of readable code via programming-by-example. This removes the need for data scientists to remember the specific functions or syntax needed for a transformation. Instead, they need only know what they want to do with the data in order to produce code.

Searching for Solutions: To alleviate recall issues, data scientists rely on web searches for solutions on websites like Stack Overflow. These searches occur because "most of the tasks are pretty standard, I expect there to be one function that solves the piece, generally in Stack Overflow, if you are able to break the problem down small enough you can find a teeny code snippet to test." (P3). Participants believed searching for these code snippets is quicker than producing the solutions themselves. This helps them reach their goal of "achieving the final result as fast as possible", so they prefer "to save time and use something existing" (P8). Unfortunately, searching for solutions can fail or increase data wrangling time depending on the domain of the task since “there are so many [web pages] and you need to pick the right one. So, it takes time to find someone who has the exact same problem that you had. Usually in 70-80% of the cases I’ve found that someone else has had the same problem, sometimes not, depends on the domain. [...] In audio [data] it’s more complicated to find someone who did something similar to what you were looking for” (P8).

WREX reduced participants’ reliance on web searches. Instead of hunting online for the right syntax or API calls, they could remain in the context of their wrangling activities and only had to provide the expected output for data transformations. Participants immediately noticed the time it took to complete the three tasks with WREX compared to doing so with a default Jupyter notebook and web search: “I super liked it, it was amazing, really quick, I didn’t have to look up or browse anything else” (P9); WREX also “avoided my back and forth from Stack Overflow.” (P12). By removing the need to search websites and code repositories, WREX allows data scientists to remain in the context of their analysis.

3.7.2 Fitting into Data Scientists’ Workflows

WREX helps address the above barriers by providing familiar interactions that reduce the need for syntax recall and code-related web searches. First, WREX’s grid felt familiar, lessening the learning curve required to perform data wrangling tasks with the system. This form of interaction was likened to “the pattern recognition that Excel has when you drag and drop it” and that WREX had a “nice free text flow” (P5). Feedback for the grid interaction was overwhelmingly positive (Table 3.4), with only minor enhancements suggested such as a right-click context menu and better horizontal scrolling.

Participants agreed that this tool fit into their workflow. They were enthusiastic about not having to leave the notebook when performing their day-to-day data wrangling tasks. By having a tool that generates wrangling code directly in their notebook, participants felt that they could easily iterate between data wrangling and analysis. Some participants reported running subsets of their data on local notebooks for exploratory analysis, but then eventually needing to export their code into production Python scripts to run in the cloud. With existing data wrangling tools, participants indicated that they would have to re-write these transformations in Python. Because Wrex already produces code, these data wrangling transformations are easy to incorporate into such scripts.

3.7.3 Data Scientists' Expectations of Synthesized Code

Readability of Synthesized Code: Participants described readability as being a critical feature of usable synthesized code. P6 wanted "to read what the code was doing and make sure it was doing what I expect it to do, in case there was an ambiguity I didn't pick up on". It is also important for collaboration "because the whole purpose for me to use Jupyter notebook is to be able to interpret things. [So], if I leave and pass on my work to someone else, they would be able to use it if they know how it is written" (P12). Participants also cited readability as an enabler for debugging and maintenance, where readable code would allow them to make small changes to the code themselves rather than provide more examples to the interactive grid. Amongst our participants, some example standards for readability were: "I would want it to be very similar to what I would expect searching Stack Overflow" (P10). Interestingly, our participants found short variable names like "String s" or "Float f" to be acceptable, as they could just rename these themselves.

Trust in Synthesized Code: The most salient method to increase trust was reading the synthesized code. Inspecting the resulting wrangled data frame is not enough, and that without readable code they "don't know what is going on there, so I don't know if I can trust it if I want to extend it to other tasks. I saw my examples were correctly transformed, but because the code is hard to read, I would not be able to trust what it is doing" (P10). Several participants noted that the best way to gain the confidence of a user in these types of tools is to "have the code be readable" (P3).

Several participants proposed alternative methods beyond inspecting the data frame and the data wrangling code to improve their trust of the system. These proposals ranged from simple summations of the resulting output, code comments, and data visualizations. Some participants desired information on any assumptions made for edge cases, or to request examples for these edge cases. These alternative affordances are important, as they could provide “a way of validating, maybe not the mechanics, the internals of it, but the output of it, would help me be confident that it did what I thought it did” (P2). That said, if Wrex did not produce readable code, some participants “would be less trustful of it” (P10).

3.8 Discussion

3.8.1 Data Scientists Need In-Situ Tools Within Their Workflow

Computational notebooks are not just for wrangling, but for the entire data analysis workflow. Thus, programming-by-example (PBE) tools that enhance the user experience at each stage of data analysis need to reside where data scientists perform these tasks: *within the notebook*. These in-situ workflows are an efficiency boost for data scientists in two ways: First, providing PBE within the notebook removes the need for users to leave their notebook and spend valuable time web searching for code solutions, as the solutions are generated based on user examples. Second, users no longer need to export their data, open an external tool, load the data into that tool, perform any data wrangling required, export the wrangled data, and reload that data back into their notebook.

Though our investigation focused on data wrangling, tools like Wrex can play a critical role at each step of a data analysis by providing unified PBE interactions. For example, future PBE tools can first ingest data to synthesize code that creates a data frame, which can then be wrangled using PBE, and finally again be used to produce code to create visualizations like histograms or other useful graphs. This provides an accessible, efficient, and powerful interface to data scientists performing data analysis, allowing them to never leave their notebooks and thus avoid context-switching costs.

In our user studies we found that data scientists were unlikely to adopt tools that required them to leave their notebook. We also witnessed participants struggle to find code online that was suitable to the task at hand. Without Wrex, they had to frequently move back and forth between web searches and their notebook as they copied and modified various code snippets. With Wrex, this frustration was removed. When PBE interactions are within the notebook, a streamlined and efficient workflow for data analysis can be realized. In sum, as long as interactions are in-notebook, familiar, and can produce readable code, data scientists are enthusiastic to adopt PBE tools; they are hesitant to do so without these features.

Also, notebooks are the ideal environment for PBE tools since program synthesis is good at generating small code snippets which is a similar granularity to existing notebook cell usage. Program synthesis also relies on user interactions to provide examples that remove ambiguity so that PBE tools can produce correct code. Notebooks already provide a platform that enables the interaction between a user and their code that is a good match for the user interaction requirements of PBE.

3.8.2 Data Scientists' Priorities for Readable Synthesized Code

Data scientists need to be able to read and comprehend the code so they can verify it is accomplishing their task. Thus, if a system synthesizes unreadable code, users have much more trouble performing verification of the output. Verifiability increases trust in the system, and gives data scientists confidence that the synthesized code handles edge cases and performs the task without errors. Readability also improves maintainability. If a data scientist wants to reuse the synthesized code elsewhere but make edits based on the context of their data, they need to be able to first understand that code.

Data scientists prioritize certain readability features over others when thinking about acceptable synthesized code. Participants did signal a need for features that increased readability like better indentation, line breaks, naming conventions, and meaningful comments in the synthesized code. Some participants desired synthesized code that followed language idioms or that “would pass a [GitHub] pull request”, but other participants saw their notebooks as exploratory code that would need to be rewritten and productionized later anyway, and instead, desired synthesized code that is brief and easy to follow. This means that the goal of synthesized code should not be to appear as if a human had written it, but to focus on having these high priority features that data scientists require.

3.8.3 Interactive Data Wrangling versus Directly Coding

In some situations, grid interactions introduced more overhead than a code-only workflow, like when the data scientist knows the exact wrangling task they want to accomplish.

When verifying the effects of data wrangling code, data scientists have certain heuristics they use. During our study, several participants noted that they check for possibly erroneous data values by generating data visualizations, printing the frequency of `NULL`s in individual columns, and checking that range of values in a column seem correct. P1, for instance. said they made sure to check that no timestamps occurred after the current date, as that could point to an issue with the data in that dataset. Data wrangling tools like Wrex could be improved by providing these heuristics to data scientists, increasing their confidence that the code is correct.

Verifying the data through an interactive grid can be burdensome for large samples, as data scientists would need to scroll through every row to check for any incorrect outputs. To alleviate this, we can suggest inputs to the user that would be beneficial to receive examples for. This “significant inputs” capability exists within the program synthesis engine, but was not surfaced in the current implementation of Wrex.

3.8.4 Alternative Interactions with Code and Data

Data scientists frequently use applications like Excel to view their data and Python IDEs to manipulate it, which make them choose between the ease of use afforded by GUIs, and the expressive flexibility afforded by programming. Wrex merges usability and flexibility by generating code through grid interactions. We found that our grid was familiar to data scientists who had used various grid-like structures before in spreadsheets. By implementing our grid in a programming environment such as Jupyter Notebooks, our system fits into data scientists' existing code-oriented workflow. Though our original aim was to help data scientists accomplish difficult data wrangling tasks, our participants found that WREX was also useful for performing simpler PBE tasks like adding or dropping a column. While we implemented our interaction with program synthesis as an interactive grid, we believe that other interactions can also synthesize readable code. For example, our study participants mentioned data summaries and visualizations as potential sources for verification of the output of data science tools. One requested feature was histograms of the initial and the updated data frame so users can take a quick glance and make sure the shape of the data makes sense. Data summaries provide ranges of values that provide potential edge cases for their code to handle, either by feeding them as examples to WREX or by modifying synthesized code to handle these cases. The insight we gleaned from this feedback is that data scientists want the freedom that comes with multiple workflows so they can choose the best interaction for each task. In future work, it is interesting to explore different surfaces for exposing PBE tools, like the visually-richer interfaces described above, while discovering and minimizing potential trade-offs in user experience.

3.8.5 Synthesized Code Makes Data Science More Accessible

Synthesizing code with PBE has the potential to make data science more accessible to people with varying levels of programming proficiency. For instance, without a tool like WREX, a data scientist in a neuroscience lab must not only become an expert on brain-related data but also in the mechanics of programming. With WREX they can not only see the final wrangled data, which speeds up their workflow, they can also study the code that performed those transformations.

Our study participants noted that WREX was useful in learning how to perform the transforms they were interested in, or even assist them in discovering different programming patterns for regular expressions. WREX also alleviates the tedium felt by data scientists having to learn new APIs and even lessens the burden of having to keep up with API updates. This also benefits polyglot programmers who might be weaker in a new language, as they can quickly get up to speed by leveraging Wrex to produce code that they can use and learn from. In the future we see potential for interactive program synthesis tools as learning instruments if they are able to synthesize readable and pedagogically-suitable code.

3.9 Conclusion: Synthesizing Readable Code

Our formative study found that professional data scientists are reluctant to use existing wrangling tools that did not fit within their notebook-based workflows. To address this gap, we developed Wrex, a notebook-based programming-by-example system that generates readable code for common data transformations. Our user study found that data scientists are significantly more effective and efficient at data wrangling with Wrex over manual programming.

In particular, users reported that the synthesis of readable code—and the transparency that code offers—was an essential requirement for supporting their data wrangling workflows.

3.10 Summary and Acknowledgements

This chapter of the dissertation presents a formative study with data scientists discover design principles for data wrangling systems. It then explains the design and implementation of Wrex, a hybrid interaction model that combines programming-by-example with readable code synthesis within the cell-based workflows in computational notebooks. It details a user study with data scientists that studies Wrex's effectiveness, which found that data scientists in the Wrex condition were significantly more effective and efficient than with manual programming. Interviews with participants to gather qualitative feedback on the effectiveness and acceptability of Wrex found that it could reduce barriers to data wrangling like searching for and recalling data transformation functions, verifying the tool is performing the task correctly by inspecting the readable Python code, and participants agreed that Wrex fit within their existing computational workflows.

Chapter 3, in full, is a reprint of the material as it appears in the proceedings of the 2020 CHI Conference on Human Factors in Computing Systems. Ian Drosos, Titus Barik, Philip Guo, Robert DeLine, and Sumit Gulwani. 2020. The dissertation author was the primary investigator and author of this paper.

Chapter 4

Streamers Teaching Programming, Art, and Gaming: Cognitive Apprenticeship, Serendipitous Teachable Moments, and Tacit Expert Knowledge

Livestreaming is now a popular way for programmers, artists, and gamers to teach their craft online. In this chapter we propose the idea that streaming can enable cognitive apprenticeship, a form of teaching where an expert works on authentic tasks while thinking aloud to explain their creative process. To understand how streamers teach in this naturalistic way, we performed a content analysis of 20 stream videos across four popular categories: web development, data science, digital art, and gaming. We discovered four kinds of serendipitous teachable moments that are reminiscent of cognitive apprenticeship: 1) creators encountered unexpected errors that led to improvised problem solving, 2) they generated improvised examples on-the-fly, 3) they sometimes went on insightful tangents, 4) they paused to give high-level advice that was contextualized within the work they were currently performing. We also found missed opportunities for additional teachable moments due to creators not being able to express their tacit (unspoken) expert knowledge because of pattern irreducibility, context dependence, and routinization. To better support livestreaming as cognitive apprenticeship, we use our study findings to propose the design of new tools for eliciting tacit knowledge and for repurposing longform streams into shorter, more targeted videos to ease viewer consumption.

4.1 Introduction

Over the past decade there has been tremendous growth in livestreaming on platforms such as Twitch and YouTube [40, 57, 130]. While many streamers produce content for entertainment (e.g., playing video games), there is an increasing number who stream for educational purposes [37, 93]. For instance, programmer Suz Hinton does weekly livestreams showing herself working on open-source software and teaching programming concepts within the context of code that she is writing [62, 63]. These streams are often archived as videos on YouTube so that others can watch later (albeit without live interactions).

We refer to these kinds of videos as *instructional workflow streams* because they teach concepts within the context of a practitioner's naturalistic workflow, such as a programmer or data scientist writing code for their work or a digital artist using Photoshop. In this chapter we propose the idea that instructional workflow streams are compelling from an educational perspective because they enable viewers to engage in a virtual form of *cognitive apprenticeship* [21, 124]: watching an expert practitioner working on their craft while thinking aloud to verbalize their creative and technical process. Cognitive apprenticeship captures an expert working on realistic tasks while "making thinking visible" [21] so that "learners can see the processes of work" [21]. Being able to virtually "look over the shoulder" of an expert at work is especially relevant in light of global COVID-19 quarantines and the increasing prevalence of remote learning [102].

To understand how streamers teach in this naturalistic way, we performed a content analysis of 20 instructional stream videos across four popular categories [40]: web development (design+programming), data science (programming), digital art, and gaming (see Figure 4.1). To our knowledge, we are the first to perform an in-depth analysis of pedagogical content within instructional stream videos; prior work on livestreaming mostly focused on motivations and challenges faced by streamers and viewers [2, 14, 15, 37, 40, 93]. We made two new discoveries that have not been detailed in prior work:

First, we discovered four kinds of *serendipitous teachable moments* that are reminiscent

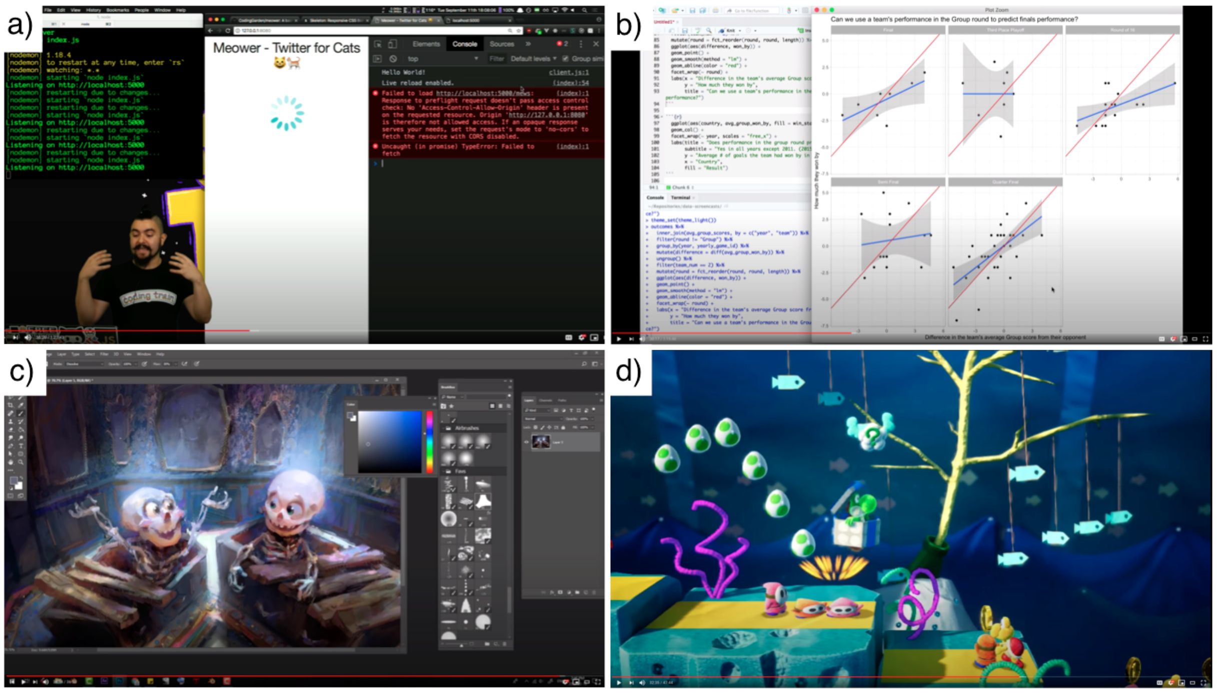

a)
b)
c)
d)
Meower - Twitter for Cats
Can we use a team's performance in the Group round to predict finals performance?

Figure 4.1. We analyzed the content of 20 archived stream videos to characterize how streamers teach authentic technical workflows across four popular domains: a) web development, b) data science, c) digital art, d) gaming.

of cognitive apprenticeship: 1) creators encountered unexpected errors that led to improvised problem solving, 2) they generated improvised examples on-the-fly, 3) they sometimes went on insightful tangents, 4) they paused to give high-level advice that was contextualized within the work they were currently performing.

Second, we found missed opportunities for additional teachable moments due to creators not being able to express their tacit (unspoken) expert knowledge [20, 115]. These arose in three ways: pattern irreducibility (cannot verbalize fine-grained details), context dependence, and routinization (actions become so routine that they are unconscious).

To better support these instructional streamers, we use our study findings to propose the design of new tools for eliciting tacit expert knowledge and for repurposing longform streams into shorter, more targeted videos to ease viewer consumption.

This chapter advances our knowledge of how practitioners use popular streaming media to teach in a naturalistic manner, which enables a novel (albeit limited) form of virtual cognitive *apprenticeship*. We envision these instructional workflow streams becoming more popular in the coming years as remote learning becomes even more widespread, spurred by our current global pandemic. Due to widespread quarantines, it is now infeasible to gather together in physical workspaces to engage in in-person cognitive apprenticeships, so virtual formats like these streams will become increasingly important for scaling up this form of hands-on experiential learning.

This chapter's contributions are:

- The idea that *instructional workflow streams* can act as a virtual form of cognitive apprenticeship [21].
- Analysis of 20 instructional workflow streams from four domains, framed through the lens of cognitive apprenticeship. We found four types of serendipitous teachable moments within these streams: improvised problem solving, generating improvised examples, insightful tangents, and contextualized high-level advice. We also found missed teachable moments due to tacit (unspoken) knowledge.
- Design ideas for specialized tools to help creators make more effective instructional workflow streams.

4.2 Methods

We conducted a qualitative study of 20 instructional workflow streams to characterize the ways in which streamers teach.

4.2.1 Video Selection

We chose four domains that span the gamut of technology-related streams that are popular on sites such as Twitch and YouTube [40]: 1) *Web Development* videos each show a programmer designing, coding, and deploying a web application, 2) *Data Science* shows a data scientist walking through their data analysis workflow using tools such as Python, R, or Excel, 3) *Digital Art* features an artist drawing in a software application using a digital tablet, 4) *Gaming* shows someone commenting on their gameplay footage to teach specific strategies. These domains cover a variety of technical skills including programming and debugging (Web Development and Data Science), open-ended creative processes (Data Science and Digital Art), manual hand dexterity (Digital Art and Gaming), and optimizing for speed (Gaming speedruns [126]).

We found a sample of videos by searching YouTube for terms such as ‘workflow’ and ‘guide’ along with keywords common to each domain. Note that even though Twitch is the most popular platform for livestreaming, many Twitch streamers archive their videos on YouTube since Twitch has limited support for long-term archiving and discoverability [7, 120].

We specifically looked for videos that came from *complete end-to-end longform workflows*, such as an artist creating a digital drawing from scratch or a web developer coding up a full-stack web application from scratch. Within each domain, we sought to find videos that varied in the types of presented content (e.g., digital art videos that show sketches, coloring, and 3D modeling). We also sought to find videos both from popular streamers (e.g., with > 100,000 subscribers) and from smaller niche ones (see ‘Subs’ column in Table 4.1).

4.2.2 Data Overview

Table 4.1 shows the 20 archived stream videos that we selected. The majority of each stream consists of the creator working on a realistic task in their domain and thinking aloud. Some are streaming their everyday work with commentary, such as this university researcher introducing his stream where he analyzed epidemiology data for an academic study (D2):

> *“So what I’m gonna do is show you what, I guess, the life of a data scientist is, which is usually not super glamorous or glorious, but I’m just gonna do it because I gotta get it done anyway. Work has to get done, so might as well have you take a look.”*

Others create a realistic example for teaching purposes rather than streaming their actual work. For instance, W3 and W4 both walk through building a simple but complete full-stack web application to teach web programming concepts.

Although most videos in Table 4.1 were unedited, four of the gaming ones (G1–G4) were

Table 4.1. The 20 instructional workflow streams that we analyzed, which span four domains: Web Development (W1–W5), Data Science (D1–D5), Digital Art (A1–A5), and Gaming (G1–G5). Each video was from a different creator; 'subs' is their current number of subscribers (k=thousand, M=million).

ID	Length	Subs	Contents
W1	2:29:06	4.3k	Coding up a personal website to show the end-to-end design process
W2	1:29:18	78k	Wireframing and coding a portfolio website using Balsamiq and HTML/CSS/JavaScript
W3	1:27:43	1.3M	Building a full-stack Twitter clone with a Node.js backend
W4	2:40:29	1.7k	Building a real-time web app using WebSockets and Twilio API [138] for SMS
W5	25:57	59k	Deploying a personal website to cloud hosting using Git and Beanstalk [145]
D1	1:15:48	11k	Analyze Women's World Cup data with R
D2	1:18:24	25k	Data analysis for an academic research project in public health
D3	1:21:49	41k	Doing a Kaggle challenge on this data set: *Titanic: Machine Learning from Disaster* [70]
D4	1:33:03	5.8k	Computational finance with Excel
D5	1:29:52	155	Handling missing data during analysis
A1	56:16	26k	Walkthrough of regimented art training process using digital pencil sketches
A2	1:25:55	51k	Digital coloring with Procreate 4 for iPad
A3	1:51:08	52k	Practice with drawing human eyes
A4	36:56	409k	Halloween painting combining 3-D (Blender) and 2-D (Photoshop)
A5	31:50	92k	Two artists walking through creating sci-fi interior scenes using SketchUp
G1	17:01	163k	Building a character in Path Of Exile
G2	35:03	1.5k	Speedrunning the Celeste game
G3	10:38	176k	Creating macros in Final Fantasy 14
G4	15:26	18k	Guide for playing Scholar character class in Final Fantasy 14
G5	47:44	4.7M	Gameplay walkthrough ('playthrough') of Yoshi's Crafted World

edited to extract highlights from longer gameplay livestreams. Those creators added voiceover commentary to explain strategies used during gameplay. Their original streams were several hours long and contained almost no talking since they were concentrated on playing the games.

4.2.3 Media Content Analysis using Cognitive Apprenticeship

We performed a media content analysis on these videos, which is a standard qualitative research technique in fields such as communications and media studies [100]. This method involves treating the media itself as the primary subject of study and interrogating its contents via a theoretical lens.

We analyzed the content within these 20 videos through the lens of *cognitive apprenticeship* [21, 124], which is a process by which novices learn from an expert practitioner by observing them at work in a naturalistic environment. Thus, we watched these videos to look for pedagogically-relevant content by adopting the role of apprentices who were virtually watching an expert at work on their craft. We found that these videos were amenable to this sort of content analysis since their creators purposely made them with educational intent in mind, so they performed think-aloud to explain their thought process.

Two members of the research team individually watched each video and read the archived livestream chats (if available). Specifically, we looked for how creators 1) articulated domain knowledge in the context of their workflow, 2) demonstrated their problem-solving heuristics, and 3) reflected on metacognitive strategies, which are the three common facets of cognitive apprenticeship [21]. We each performed open coding and met regularly to reconcile our codes when there were disagreements. We iteratively merged our observations using an inductive analysis approach to build a grounded theory [23] of how these videos embody aspects of cognitive apprenticeship. We decided to stop after 20 videos since we felt we had reached a reasonable qualitative saturation point [123] when the same themes kept re-appearing in multiple videos and it became harder to find distinctively new themes.

4.2.4 Study Design Limitations

Although we strove to include videos from a variety of technical domains, any sample is limited and not representative of all instructional workflow streams. Since we sampled only five videos from each domain, we cannot draw any definitive conclusions about how experts teach specific skills within each domain; thus, our findings are about *high-level patterns* that we observed across multiple domains. Also, we did not analyze videos in physically-based domains such as woodworking, makeup, fashion, sports, or playing musical instruments.

Since our media content analysis was qualitative, it is subject to our own interpretations as to, for instance, what counts as a pedagogically-meaningful moment. To get multiple perspectives, two members of the research team independently watched each video, met to reconcile our notes, and re-watched portions together. However, we may have still missed certain types of events. We drew on our own personal experiences as both creators and consumers of instructional videos in these domains, but our expertise is necessarily limited. To minimize interpretation biases, throughout our findings we report direct quotes that creators spoke in their videos.

Finally, we did not interview creators about their workflows and instead relied only on content analysis of their videos. Thus, we can only infer intent based on what we directly see in videos, so those might be subject to our own interpretation biases. Note that even if we had interviewed these creators, they might themselves be subject to cognitive biases (e.g., hindsight bias) when reflecting on videos that they recorded months or years earlier. Thus, that is why media content analysis is standard practice in media studies, since the media itself is considered the primary data source [100]. Similarly, we did not interview audience members of livestreams since we watched only archived videos of streams; thus, we cannot make any claims about whether streams helped viewers learn.

4.3 Serendipitous Teachable Moments

During our video content analysis we found general forms of technical knowledge transfer that have already been reported in prior work: For instance, Alaboudi and LaToza [2] described how streams start with an introduction and outline of the planned task, followed by most of the time working on the task while thinking aloud, and concluding with a live demo of the working artifact (e.g., a web application). In addition, streamers and viewers exchanged technical knowledge such as programming tips and keyboard shortcuts [2, 37, 56].

In addition to these high-level observations that corroborate prior work, our study made two new sets of discoveries, which we will describe in this section (Section 4.3: serendipitous teachable moments) and the next (Section 4.4: missed teachable moments due to tacit knowledge).

First, since instructional workflow streams are recorded naturalistically while a streamer is working on real tasks, they contain lots of improvised content that may not appear in a lecture or tutorial video. We observed that some of this improvised content may actually have educational value since it emulates what happens during cognitive apprenticeship [22] as viewers watch an expert at work on real tasks. We call these *serendipitous teachable moments* since they are unplanned moments when the creator teaches something new and unexpected. We identified four kinds of serendipitous teachable moments: 1) improvised problem solving due to unexpected errors, 2) generating improvised examples, 3) insightful tangents, 4) contextualized high-level advice.

4.3.1 Improvised Problem Solving Due to Unexpected Errors

Some kinds of unexpected errors that creators encountered while working turned out to be sources of serendipitous teachable moments. Most notably, errors gave creators a chance to do improvised problem solving. This lets viewers see an expert genuinely struggle to solve problems that they were not prepared to encounter. This level of transparency simulates aspects of cognitive apprenticeship: seeing experts debug and troubleshoot in authentic work scenarios.

This impromptu problem-solving process was most visible in programming streams (data science and web development) where creators had to engage in information foraging and debugging to solve their coding problems while thinking aloud. They typically did web searches, visited Stack Overflow and API documentation, copied code snippets into their IDE, and tested them out. The longform video format enabled them to take their time and show their entire process, even when it took unexpected turns. For instance, W5 encountered an error while deploying a live website to the cloud and said, *"Now the .htaccess file is screwed up, we might as well fix that too ... we just gotta ... gotta do it. We got to get our local environment set up properly here."*

Seeing experts make and recover from mistakes can help learners get a sense of how creative work can be messy; one benefit of cognitive apprenticeship is that "learners can see the processes of work" [21]. Some un-did their mistakes and re-attempted, such as artist A1 acknowledging their mistake verbally then re-drawing a sketch layer: *"[laughs and deletes layer] let me do a better job with it [re-makes layer]."* A2 admits that they are still learning to improve their coloring technique, tests out several color brushes in the painting app to see their effects, hits undo repeatedly, then tries again until the desired effect is achieved. Chen et al. [15] corroborated these observations via interviews with streamers and viewers.

Errors can also make creators more relatable since viewers can see that experts also make mistakes as part of their normal workflow. At the end of their stream, W5 apologetically said, *"So sorry that was such a mess, you know in a perfect world I'd redo it and not make any mistakes and make this perfect workflow thing."* But the top-voted YouTube viewer comment below that video was: *"Watching you hilariously deal with typical everyday issues is the best part."*

Finally, the audience can participate in collaborative problem-solving with the streamer via the chat interface. For instance, when G3 is creating a custom gameplay macro, the audience notices some typos, which he fixes right away; the audience shares in the excitement of the working macro and further encourages the streamer. And in W2, W3, and W4 the audience helps the creator to debug their web application code via text chat. The most engaged participation came in D2, where the data scientist asked the live audience to look at the data set together to try to spot anomalies that might affect the analysis. (Note that even though we did not watch these streams live as they happened, chat logs are sometimes archived with videos and can be replayed in sync.) Several studies have documented how viewers can help programmers debug via chat [2, 37], but our findings show how collaborative problem-solving goes beyond debugging.

4.3.2 Generating Improvised Examples

Creators all work through a long-running main task in their videos, but they sometimes improvise to generate additional examples that are not part of their plan. These improvised examples can also serve as serendipitous teachable moments.

When explaining a concept as they are working, some creators make up mini-examples to demonstrate it more concretely. For instance, artist A1 spontaneously sketched a few *bad* pencil drawings when talking about common pitfalls that novices face. And when demonstrating a certain detail about drawing irises in eyes, artist A3 decides to zoom the camera in on his eyes and moves his eyes in different directions to provide examples in the physical world. (We inferred that these examples were improvised and unplanned since they interrupted the artists' flow on their main task, which was making progress on creating a piece of artwork.)

Creator mistakes can also lead to improvised examples. For instance, D1 was confused that a data set did not contain some data that it ought to have, so he wrote web scraping code to grab that data from Wikipedia. After he finished, he realized that he was mistaken: the data he wanted was there all along in another table in his data set, so his improvised web scraping code was unnecessary. But he still left that code in his R script to serve as a self-contained example to demonstrate web scraping. When he realized his mistake, he said in a surprised tone of voice, *"Oh wow! Whoops that was fun but I think I totally missed [this other data table], but I'm gonna leave this code in as a web scraping example."*

Questions that livestream viewers ask in the chat can also inspire creators to generate new improvised examples. This technique exemplifies “show, don’t tell” since creators can craft a small example to respond to questions rather than just verbally answering the question. For instance, a viewer notices data scientist D2 running a `pushd` command in the macOS terminal and asks a chat question about that command; D2 explains by improvising an example of using a pair of `pushd` and `popd` commands on the terminal to navigate through a Unix-like filesystem hierarchy on macOS. Web developers W3 and W4 implement unplanned features in their respective web applications due to viewer requests in the chat; those can serve as additional example code. W4 even deploys an improvised feature to the web and lets viewers test it out.

4.3.3 Insightful Tangents

Creators sometimes go off on tangents that are unrelated to the main focus of the workflow they are demonstrating. For instance, while editing code in the Vim text editor, W4 reflexively does a quick keyboard shortcut to delete an entire block of code within parentheses. He realizes that this may be interesting to some viewers, so he goes on a brief tangent to explain how that Vim keyboard shortcut works (unrelated to his main task). Similarly, A5 consists of two artists talking about their workflow, and one of them goes on a tangent about a line style tool in the SketchUp app by demonstrating the effects of selecting different styles on their current drawing. We infer the presence of tangents in these videos by creators saying phrases like *“oh, by the way ...”* or suddenly changing topics while in the midst of working on their main task.

Some tangents have a more direct pedagogical purpose. For instance, D1 noticed that one of the Women’s World Cup soccer game scores was 13-0, which seemed unusual, so he searched for the news on Google News to make sure that the score was accurate; then he spent a bit of time browsing news articles containing relevant sports statistics. While somewhat tangential, this action demonstrates a good expert habit of checking for data quality when outliers are present.

Many tangents also arose from audience questions in livestream chats. For instance, in programming streams the audience may ask about what certain functions or API calls in the code do. When the creator tries to answer, they sometimes go off on a tangent to search the web for related resources to show the audience. More complex kinds of questions include asking about comparing alternative technologies, such as a chat question asking W4 the pros and cons of using Ajax versus WebSockets for real-time apps and asking W3 about alternative ways of writing UI event handlers in JavaScript.

Some creators mentioned having trouble balancing the trade-off between making forward progress on their work and answering audience chat questions, which could lead to off-topic tangents. For instance, artist A2 told their audience: *"All right, I'll try to keep one eye on the chat as well, but if I miss things for a while, I'm sorry. I can't draw and look at that at the same time."* W3 addressed this issue by having a second streamer serving as a co-host to curate selected questions from the chat and pass them along during breaks.

4.3.4 Contextualized High-Level Advice

We also noticed that creators periodically pause their work to give high-level advice about their craft, which can help viewers zoom out and get a more holistic view. For instance, artist A4 took a short break while working and mentioned:

> *"Just a quick anecdote. I remember when I was younger and just getting into this stuff, I would email artists that I like and ask them what their process was [...] what I was looking for then was a path that I could also follow [...] but what I found was that process is not about step 1, step 2, step 3. Process is more about which fundamentals of art you need to deploy when. Using this piece as an example ..."*

... then goes on to show how the current Photoshop image they are working on exemplifies this idea of learning art fundamentals. This example shows how their high-level advice is *contextualized* within the setting of their current work.

Similarly, as they were working on drawing human forms, A3 and A4 both gave the advice to dive deep into understanding human anatomy rather than just following superficial tutorials: *"[My illustration] skills were built over, and still are being built over years of gesture drawing, understanding light and shadow, with technical studies, you know boring stuff like* *studying anatomy out of textbooks."* (A4)

After spending some time investigating missing data, D2 pauses to tell the livestream audience why it is important to spend the time upfront doing this tedious data science task:

> *"Like Elizabeth said, it's painful to chase N's [finding out why some data is missing] but really that's where I discover all the basic problems. I could have just gone into the data, just start reading it in, and you would have been like at the point of publishing a paper before you realized that maybe you were missing 20 million people [in your analysis], which hasn't happened yet and I don't want it to happen."*

Aside from general technical advice, creators also mentioned the importance of developing certain *expert mindsets*. For example, A2 advised learners that they *"should always be thinking in 3-D all the time, which takes a lot of practice"*. A3 said that to get good at art, one must always be willing to constructively critique one's own prior work. D1 mentioned the importance of domain-specific knowledge in becoming an expert data scientist – i.e., that simply knowing tools is not enough. G4 talks in depth about being proactive versus reactive during gameplay, and the importance of efficient in-game resource management. W1 mentioned how experts often look for web design inspiration from other websites.

Another kind of high-level advice is experts commiserating with novices about common struggles. For instance, A3 showed prior bad examples of their own face drawings to discuss how it is normal for novices to suffer from "same face syndrome" [118] (i.e., drawing different characters with essentially the same face). G3 reflects on aspects that he dislikes about gameplay macros that novices create but reassures the audience that he also made those mistakes when he was new: *"I totally understand how new players think these things could help them when it actually hampers them."*

Finally, the live audience sometimes inspired the creator to engage in higher-level reflections in response to chat questions. For instance, a viewer asked A3 about manga-style drawings of faces (even though A3 was not drawing manga), which led to a high-level discussion about the philosophy of manga art.

4.4 Missed teachable moments: Tacit Knowledge

Even though we observed many instances of serendipitous teachable moments, we also found missed opportunities for additional teachable moments. Some of these occurrences may be due to creators being unable to communicate their *tacit knowledge* [20, 64, 115] while streaming. As Polanyi states in his definition of tacit knowledge, *"We can know more than we can tell"* [115]. Experts are often not able to verbalize their expertise since the complex actions they take feel intuitive and unconscious to them. Elaborating on that definition, Horvath et al. [64] identified three reasons why certain knowledge remains tacit: 1) pattern irreducibility, 2) context dependence, and 3) routinization. We use these reasons to classify potential[1] instances of tacit knowledge in the 20 streams we studied:

4.4.1 Pattern Irreducibility

Horvath et al. define *pattern irreducibility* as the phenomenon that "some knowledge concerns information patterns that cannot be reduced to rules or generalizations [...] such configurations may be easier to recognize than to define concisely." [64] We noticed this when D1 coded up a few exploratory data visualizations to get a better "feel" for the shape of the Women's World Cup data set after looking at some of the raw data tables. He quickly went through a few visualizations without explaining his rationale for picking those ones out of the vast array of possible visualizations and parameter settings he could have chosen. This may be an instance of pattern irreducibility since it is easy for an expert data scientist to recognize certain data properties and intuit what visualizations may be the most fruitful to try in each scenario, but it is hard to verbalize those intuitions into hard-and-fast rules for novices to follow.

[1] We cannot definitively tell whether these instances were due to tacit knowledge since we did not directly interact with creators live. But even if we had, by definition it is hard for experts to know what kinds of knowledge they cannot verbalize, so we would still need to infer using our own judgment.

4.4.2 Context Dependence

Some expert knowledge remains unspoken because it is highly context-dependent, which makes it hard to verbalize as generally-applicable advice. Thus, when asked whether to apply certain strategies in a given scenario, experts often say something like *"well, it depends."* This came up frequently in gaming videos since different players' gameplay contexts may vary greatly even if they are playing the same level. For instance, G4 mentioned that the strategy they show on-screen is very situationally-dependent and unlikely to be a universal rule that all players should follow. G2 even admits that it is hard to teach someone how to succeed in a particular boss fight in the game except for "winging it" based on the situation:

> *"There's honestly not much I can say about this fight as I found a lot of the movement consists of what you think works best [...] There's really not just one set of movement I do in this fight, a lot of it just turns into winging it so I don't die."* (G2)

4.4.3 Routinization

A third type of tacit knowledge refers to actions that become so routine that experts perform them unconsciously. This came up most frequently in the art and gaming domains, which both involve large amounts of manual dexterity and muscle memory that can be hard to put into words. For instance, A3 tells viewers that while there are many techniques for drawing the human eye, what they suggest most strongly is to simply practice more since that is the only way to build muscle memory: *"If you want to get better at drawing eyes, draw 100 of them."* Gaming also requires mastering muscle memory due to the rapid pace and precision required to enter controller inputs to successfully navigate the game. When teaching speedrunning [126] (the act of getting through a game as quickly as possible), G2 says *"the first maze in this section is really difficult to do quickly, so just use dashes PROPERLY and get through it as fast as you can."* G2 does not explain what it means to use dashes (a type of in-game movement) "properly" but repeats throughout the video that practicing is the only way to develop muscle memory for mastering dashes.

4.5 Discussion

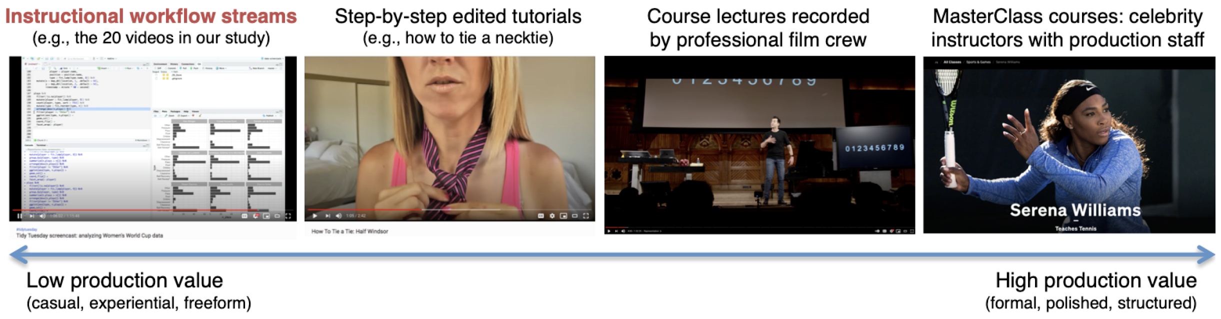

Instructional workflow streams
(e.g., the 20 videos in our study)
Step-by-step edited tutorials
(e.g., how to tie a necktie)
Course lectures recorded
by professional film crew
MasterClass courses: celebrity
instructors with production staff
Serena Williams
Teaches Tennis
Low production value
(casual, experiential, freeform)
High production value
(formal, polished, structured)

Figure 4.2. Common types of instructional videos on a spectrum from low to high production value, with the instructional workflow streams we studied at the left and professionally-produced course videos at the right.

Through this qualitative study we have demonstrated that instructional workflow streams can scale up a limited form of cognitive apprenticeship by offering viewers a glimpse into how experts perform realistic tasks across several domains. Of course, viewers do not get to have rich interactions with experts (aside from limited text chats in livestreams) or to try out the tasks themselves and get feedback on their technique, both of which are required for true cognitive apprenticeship [22]. But despite these limitations, we believe that this emerging class of videos can complement the more traditional formats of lecture and tutorial videos commonly found in MOOCs, online courses, and throughout YouTube. Thus, they represent a key step toward scaling up more experiential forms of learning.

We now reflect on our study findings by showing how instructional workflow streams fit into the broader landscape of instructional videos and how we can design tools to support this increasingly-popular medium.

4.5.1 The Production Value Spectrum of Instructional Videos

How do instructional workflow streams relate to other kinds of instructional videos that are now pervasive online? One useful way to categorize videos is based on the concept of *production value*, which refers to the time, money, personnel, and other resources required to create a video [55]. Thus, in Figure 4.2 we laid out various types of instructional videos along a spectrum from low to high production value. On the left end are the instructional workflow streams in our study. Moving to the right we see step-by-step tutorials (e.g., how to tie a necktie) that are planned, rehearsed, and edited, which result in higher production value; but those can still be produced by one person at home. Still farther to the right are more highly-polished videos captured from university lectures or recorded for MOOCs in dedicated studios. And on the very right are videos for high-end platforms like MasterClass (e.g., Serena Williams teaches tennis [104]), recorded in movie studios with a professional staff to manage scripts, lighting, set design, cameras, directing, and editing.

Despite their lower production values, we believe instructional workflow streams have several pedagogical benefits over higher-production-value videos: Most notably, their naturalistic longform nature lets creators take the time to demonstrate what is truly involved in completing real tasks. Mistakes, improvising on-the-fly, tangents, and dealing with minutiae are all realistic parts of the process that viewers now get to experience and learn from. In contrast, in a more “sterile” recording environment for high-production-value videos (e.g., for MOOCs or MasterClass), many of these seemingly-extraneous moments either never get captured or get edited out in post-production to make videos seem more polished. Guo et al. [55] found some early evidence that MOOC videos with lower production values actually get *higher engagement* possibly because they feel more relatable; instructional workflow streams may also benefit from this relatability effect, but we have not studied viewer engagement yet.

The low barrier to entry to creating instructional workflow streams also makes it easier to disseminate long-tail niche expertise: Any practitioner can stream their workflow to teach others without needing to become adept at writing scripts, directing, or editing videos; Chen et al. report that streamers mentioned how it is far easier to start a livestream than to film, and edit a formal tutorial video [15]. This approachability can also result in more people producing niche content to teach more obscure topics that do not have enough critical mass to warrant creating online courses.

From a viewer's perspective, instructional workflow streams can serve as a low-fidelity proxy for cognitive apprenticeships [21] by enabling novices to virtually sit beside an expert and observe them at work. In addition to seeing technical workflow details, viewers can also get a sense of expert mindsets, intuitions, metacognition, and higher-level reflections. This lets them engage in a form of experiential learning [80] of authentic work practices. This more informal kind of learning material contrasts with the more targeted content that lecture and tutorial videos offer. For instance, a web programming tutorial may teach how to code up a specific web API call in isolation, but that same content presented within instructional workflow streams can show the broader context of how that API is being used in real-world web development tasks.

That said, the disadvantage of instructional workflow streams is their lower signal-to-noise ratio compared to more highly-produced videos with carefully-scripted scenes and tight editing. As a result, casual viewers may be unwilling to watch through many minutes or hours of raw stream footage.

In sum, we believe that videos along the entire spectrum of Figure 4.2 play complementary roles: low-production-value streams on the left end can be valuable for experiential learning and "walking in the shoes" of an expert; mid-production-value step-by-step tutorials can be useful for targeted skills training (e.g., how to hit a backhand in tennis); and high-production-value lecture videos can generate excitement and inspiration (e.g., watching stars like Serena Williams teach tennis [104]).

4.5.2 Implications for Tool Design

To our knowledge, there are few specialized tools to support creators in making instructional workflow streams. They currently use general-purpose livestreaming tools such as OBS [1] and then post their content on platforms such as Twitch and YouTube. Based on our study findings, how can we design new tools to support creators of instructional workflow streams?

Creating more teachable moments by eliciting tacit knowledge

Our study findings showed that serendipitous teachable moments such as tangents, mistakes, and improvised examples may have instructional value. But there are missed opportunities for even more teachable moments due to experts being unable to verbalize certain forms of tacit knowledge. Although by definition much of that knowledge is hard to express, prior work in educational psychology has found that it is possible to elicit some of that knowledge with prompts (i.e., asking experts to self-reflect at salient times) [20, 87].

We envision a tool that monitors workflow activities within a given application (e.g., Photoshop, Jupyter Notebooks, a programming IDE) and asks reflective prompts during critical incidents. For instance, if the tool detects a rapid series of undo-redo actions in Photoshop, it could prompt the creator to reflect about why they are thrashing back-and-forth in quick succession. Although a tool cannot directly extract tacit knowledge from the creator's mind, by prompting them to reflect at specific moments, the creator may verbalize some kinds of knowledge that would otherwise be left unspoken. If the creator is too busy working to self-reflect at the prompted time, the tool could save a short screen recording and give a reflection reminder later after they finish the work session.

This tool could also prompt the livestream audience to ask meaningful questions at certain times. In the rapid undo-redo example, instead of popping up a general "why are you doing this?" prompt, the tool could remind viewers to ask the creator a question related to that incident. Viewers have more context and can ask more directed questions like, "Hey I noticed you often undo-redo when using the paintbrush tool to try out different variations, what's the reasoning behind this strategy?" If the creator is busy working and does not want to get distracted, then other viewers can contribute their ideas.

Repurposing longform streams into shorter tutorial videos

The aforementioned elicitation tool can enrich the pedagogical content of instructional workflow streams by getting creators to reflect more deeply on their work. However, we also acknowledge that many viewers still prefer to watch short edited tutorials focused on a specific goal rather than sitting through an hours-long workflow stream. This constraint inspired us to ask: How can tools better support creators in *repurposing* their existing streams into edited tutorials?

We envision tool support within streaming software that allows the creator and their audience to tag activity boundaries in near-real-time, similar to some of the features of StreamWiki [93]. Consider the moment when data scientist D1 realizes he needs to write some web scraping code because he could not find the data he needed in the given data set. At that moment, he could tag the start of a "web scraping" activity; his live audience could also tag it via a chat macro while watching him. More likely, though, he would be busy focusing on the web scraping task and not want to stop to tag it until he was finished. At that point, he could mark the end of the activity and rewind the video for a few minutes to retroactively mark the start time. To crowdsource this effort, a livestream audience could also do this as he continued working. Note that people could always watch the completed video afterward to tag it later, using systems like Crowdy [142] or ToolScape [77], but we believe it is vital to do the tagging live like StreamWiki [93] since the workflow context is in everyone's heads at that moment so there is no need for extra post-production time.

Manual tagging is a general-purpose solution, but it can be complemented with semi-automatic stream segmentation techniques implemented for specific applications such as those proposed by Fraser et al. [39]. For instance, by monitoring event logs of an application (e.g., Photoshop), a tool can potentially infer activity boundaries based on event types (e.g., creating a new layer, saving a file) then present those to the creator and audience for real-time tagging.

Once a workflow stream is tagged with activity boundaries and labels, creators can more easily edit it into shorter tutorial videos, and the tool can overlay step-by-step subgoal labels [142] atop the videos as text annotations. This enables creators to get the benefits of both livestreaming and edited tutorials: it is fast and easy for them to start a stream, then it takes minimal editing to produce shorter tutorial videos.

Improved viewer interfaces for instructional workflow streams

Finally, to close the loop we must ultimately focus on learners. Thus, it is important to develop better interfaces for viewing instructional workflow streams. One possible line of work, inspired by systems such as Chronicle [47] and Pause-and-Play [116], is to embed a video player into a specific software application (e.g., Photoshop, a programming IDE). The original recording can then track application-specific timestamped actions (e.g., tool selections in Photoshop) so that the player can sync them up with the app. That way, viewers can try following along with what the expert is demonstrating in their videos, which simulates how learners mimic the expert's actions during in-person cognitive apprenticeship [22].

Another line of work here could involve social video viewing interfaces, inspired by "viewing parties" that learners now hold over Zoom [4] to watch pre-recorded lecture videos together and discuss their contents. If viewing parties were facilitated by an enhanced video player along with screensharing, then a group of learners could collectively mimic what they see the creator doing in the video and then give real-time feedback to one another. These viewing parties could also be supervised by more advanced learners serving in a role like volunteer Community TAs in MOOCs [30]. We hope social viewing interfaces can facilitate some of the interpersonal interactions that are present during in-person cognitive apprenticeship.

Toward a video-based workflow graph viewer

Although the activity tagging technique above can help creators repurpose existing instructional workflow streams into edited tutorial videos, we can go one step farther. Our study found that these streams contained many tangents, mistakes, backtracking, and improvised examples; in other words, real workflows are not simply a straight-line path from start to finish. This is especially the case in exploratory data analysis or design prototyping, which are domains where non-linear branching is a natural part of the creative process.

We envision a tool where the creator can take a set of tagged activities and arrange them in a *workflow graph* [132] showing the non-linear paths and backtracking throughout their stream. In the example of D1 writing web scraping code, that was actually a tangent since he later realized he did not need to write that code, so he backtracked and instead grabbed that data from another table in his data set. That could be represented as a dead-end branch in the graph.

Once such a graph is built, viewers can navigate it via an interactive visual interface where they watch segmented video clips of the activities represented in each graph node. That way, they can have a "Choose Your Own Adventure" experience. For instance, they can watch the video clips that comprise the main "critical path" through the graph from start to successful finish or explore clips of tangents represented by side branches (e.g., the web scraping tangent).

Creators can also use this same interface to "compile" paths through the graph into secondary videos. For instance, a walk through the main critical path may produce a workflow video that is completely free of tangents, mistakes, or backtracking (i.e., only showing successful actions from beginning to end). In contrast, a walk through only dead-end branches can be compiled into "outtakes" video clips that show the creator making and recovering from mistakes. With some added post-hoc creator commentary, these outtakes can make for good postmortems or lessons-learned clips.

4.6 Conclusion: Cognitive Apprenticeship@Scale

Over the past decade the volume of online learning resources has grown so vast to the point that a large array of domain-specific knowledge is freely available online. Novices can now pick up tons of facts and basic skills on-demand. However, ***what is much harder for novices to learn are authentic and contextualized expert work practices that cannot be taught as easily in textbooks or tutorial websites***.

This chapter demonstrates that instructional workflow streams are a promising medium for transmitting such experiential knowledge via a virtual form of cognitive apprenticeship. Our analysis of 20 instructional workflow streams shows that this medium can foster serendipitous teachable moments such as improvised problem solving, generating improvised examples, insightful tangents, and contextualized high-level advice.

In sum, we view this study as a first step toward creating tools to help technical practitioners such as programmers, scientists, and digital artists to share their workflows more effectively. To truly scale up virtual cognitive apprenticeship, though, the next step is to close the loop by developing tools to enable learners to follow along with these expert actions, try variations on their own, self-reflect, and get real-time feedback from mentors. We envision a future where novices can learn from authentic expert workflows as easily as they can now look up a vast array of basic knowledge online. The findings in this study point to early steps toward that long-term vision.

4.7 Summary and Acknowledgments

This chapter of the dissertation investigates the current use of livestreaming to teach cognitively complex skills like data science, programming, art, and gaming in a more naturalistic way. It presents the analysis of 20 videos categorized as instructional workflow streams, which show the longform workflows of these experts to be presented to novices as learning content. It proposes the idea that this type of streaming can enable a virtual form of cognitive apprenticeship through working on real tasks and thinking aloud about the process of being in a workflow. The analysis in this chapter found four different kinds of serendipitous teachable moments that lend themselves to cognitive apprenticeship, like improvised problem solving, improvised example creation, insightful tangents, and contextualized high-level advice. There were also many occurances of missed teachable moments, which could be due to tacit knowledge and issues with conveying that knowledge. The chapter ends with the proposal of novel tools that can elicit this tacit knowledge to enable these streams to become more effective at cognitive apprenticeship and leveraging longer streams into more succinct and digestible tutorials.

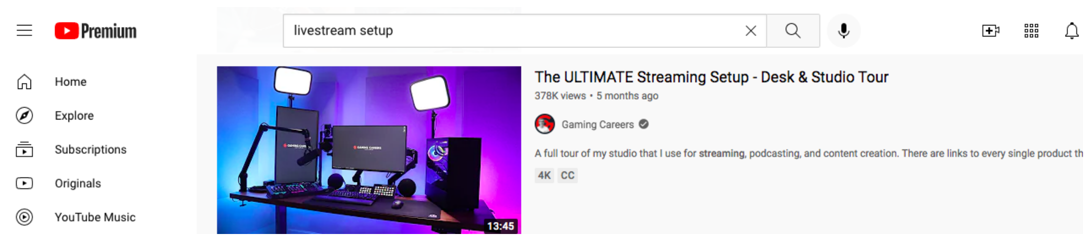
Chapter 4, in part, is a reprint of the material as it appears in the proceedings of the 2021 Symposium on Visual Languages and Human-Centric Computing. Ian Drosos and Philip Guo.

2021. The dissertation author was the primary investigator and author of this paper.

Chapter 5

The Design Space of Livestreaming Equipment Setups: Tradeoffs, Challenges, and Opportunities

Livestreaming has grown popular in recent years, with millions of people broadcasting themselves making digital art, playing games, programming, and doing other activities on sites like Twitch and YouTube. While many researchers have studied the actions of both streamers and their viewers, to our knowledge there has been no comprehensive analysis of the actual hardware and software equipment used in livestreaming. In this chapter, we present a holistic overview of modern livestreaming equipment in 2022 by analyzing 40 videos where streamers talk about various aspects of their setups. We categorized their equipment choices into a design space with ten dimensions: computer, software, stream control, encoding, cameras, lighting, video accessories, microphones, audio mixers, and audio accessories. We found that each streamer must make tradeoffs between lower- and higher-fidelity options within each dimension. Our design space analysis can inform ideas for future streaming support tools and, more broadly, tools for remote collaboration and learning via live video. As more of us work and learn online, we are in essence becoming amateur livestreamers, so understanding how professional streamers use their equipment to effectively engage their audiences might help us also engage better with our coworkers and classmates.



The ULTIMATE Streaming Setup - Desk & Studio Tour

A full tour of my studio that I use for **streaming**, podcasting, and content creation. There are links to every single product th

Figure 5.1. Example of a livestream setup video on YouTube. In this chapter we survey the landscape of livestreaming equipment setups.

5.1 Introduction

Millions of people now use platforms like Twitch and YouTube to broadcast themselves live as they are playing games [86, 101, 113, 126, 131], making digital art [40], programming [2, 15, 36, 37], and doing other activities. Since these people are acting as *single-person live video production crews*, a big challenge for aspiring livestreamers is figuring out what sorts of equipment they need in order to produce a compelling stream given their budget constraints. Some streamers give their personal tips by making 'livestream setup tour' videos (see Figure 5.1). However, these walkthroughs each cover only a limited set of options so it is hard for novices to understand the full range of equipment possibilities.

To provide a broad overview of the current state of practice, we present (to our knowledge) the first comprehensive survey of livestreaming equipment setups. We performed a content analysis of 40 livestreaming setup walkthrough and tutorial videos (e.g., Figure 5.1) to characterize the types of hardware and software that streamers recommend and the design tradeoffs of each. We synthesized these findings into a *design space of livestreaming equipment* that captures the main dimensions of variation in streamers' setups. Figure 5.2 shows our design space, which contains ten dimensions arranged into three groups: broadcasting, video, and audio equipment. Each streamer can choose options within each dimension according to fidelity, which usually correlates with monetary cost. For instance, they can repurpose (re-use) an old computer as their streaming PC, build a single high-end PC, or build an expensive multi-PC setup.

Throughout this analysis we discovered common challenges that streamers face in

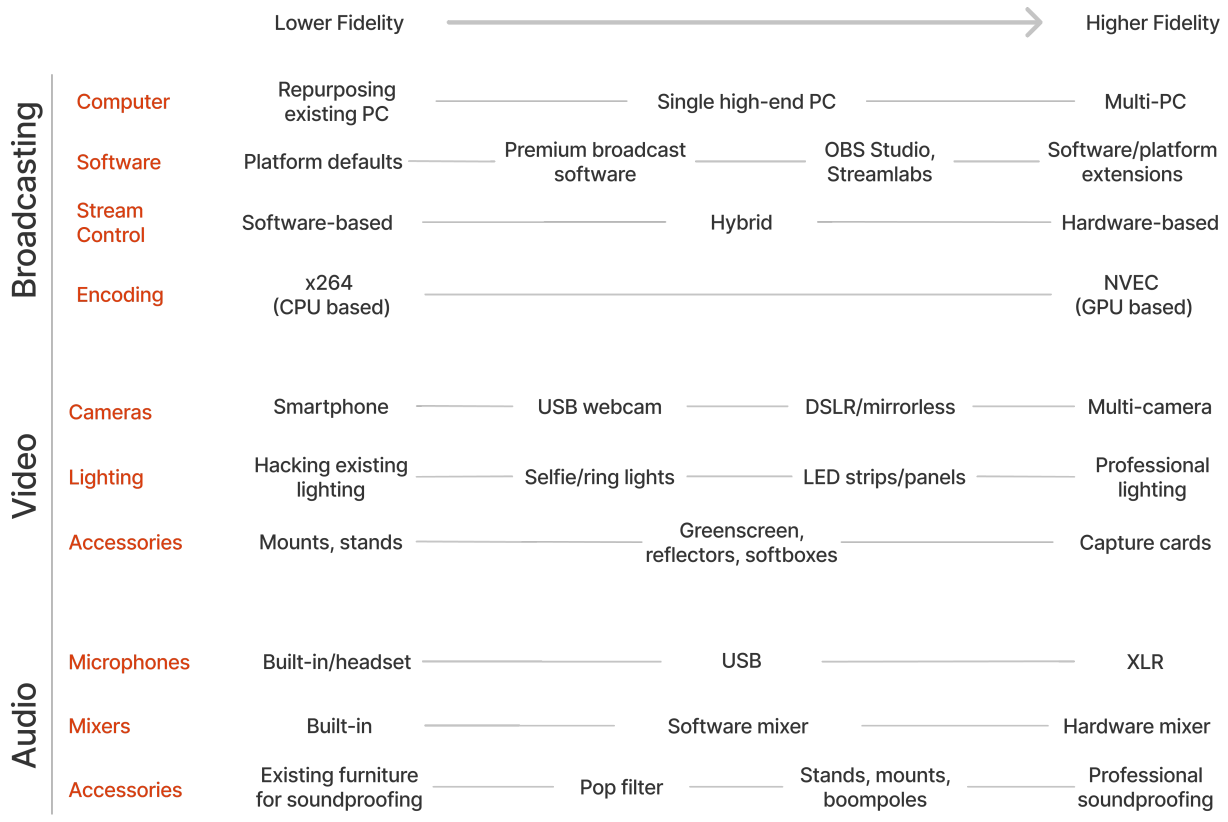


Figure 5.2. We analyzed 40 streaming setup videos to synthesize a design space of livestreaming equipment. Our design space captures ten dimensions of variation in equipment choices. Each dimension offers common options along a spectrum from low- to high-fidelity.

integrating separate components together into a unified setup, managing the logistical complexity of equipment while broadcasting in real-time, and making decisions about how to incrementally upgrade equipment without getting overwhelmed. We propose ideas for future tools that can help streamers manage technical logistics so that they can focus on the actual craft of livestreaming.

Our findings contribute to the growing body of research on livestreaming, which complements prior studies that have so far focused on either streamers or their viewers (see Section 2.2.1); in contrast, our study is the first to focus on analyzing the actual equipment used to stream, which has implications for technical systems design and integration.

More broadly, our findings have implications beyond livestreaming since the global COVID-19 pandemic has forced millions of people to work and learn remotely via live video. In essence, many of us are now amateur livestreamers as we hold more classes and work-related meetings via videoconferencing. The lessons we learn from popular modern-day streamers can potentially inform the future of remote collaboration and learning tools.

In sum, the contributions of this chapter are:

- A design space of livestreaming equipment, derived from content analysis of 40 streaming setup/tutorial videos.
- A discussion of challenges that streamers face when integrating, managing, and upgrading their equipment.

5.2 Methods

We surveyed the current state of practice in terms of the kinds of hardware and software equipment that livestreamers use. Since streamers are most adept at expressing themselves via video, we found that searching for information online about their setups yielded mostly video results (rather than, say, blog posts). We noticed that popular streamers like to post "walkthrough of my livestreaming setup" types of videos to engage with their fans who are curious about what equipment they use. Many videos we collected for our design space analysis came from this genre (see Table 5.1).

Table 5.1. The 40 livestreaming setup and tutorial videos used in our qualitative content analysis. 'yt=' is the video's YouTube ID.

ID	Length	Summary of contents
V01	13:44	Tour of a gaming content creator's high-fidelity streaming setup (yt=UrZZoDw9Yfg)
V02	07:24	Budget streaming setup with repurposing existing furniture and tools (yt=L6ZJaKqALgM)
V03	11:08	Walkthrough of streaming setup for a software engineering content creator (yt=3Zd9c-cZ5eE)
V04	13:32	Tutorial for how to stream on a gaming PC with a larger budget (yt=xcVSxchn0uM)
V05	04:42	Setting up audio for game streaming and the equipment recommended (yt=zRw1FZPrQao)
V06	18:00	Streamlabs software setup tutorial for beginner streamers (yt=pY6nhTzc85s)
V07	18:46	Building a PC streaming setup and information about video encoders (yt=Ai3nnhSIXec)
V08	20:02	Walkthrough of a PC gaming and streaming setup (yt=pvdtUSde3nw)
V09	08:24	Hardware accessories that are useful for livestreaming (yt=Y02wZGfO6Vk)
V10	04:49	Quick video about getting a stream working using OBS Studio (yt=wt-ac45JQaU)
V11	17:16	Installing a streaming setup with soundproofing in a new house (yt=ED6DWfpaGIk)
V12	10:10	Guide for setting up and using a Stream Deck [32] controller hardware (yt=MQxmuwuHJo0)
V13	14:55	Educational tech streamer going over their gear and discussing alternatives (yt=N173ajQi3X4)
V14	23:23	Guide for setting up and configuring a dual-PC stream (yt=ajSxWGCDgqM)
V15	12:04	A wide range of livestreaming equipment from beginner to pro level (yt=niP7l_F5pOU)
V16	12:10	Streaming setup tour before the streamer moves to a new studio (yt=_quSTWJx-OU)
V17	18:27	Pros and cons of using a video switcher interface for streaming (yt=KoZwgvudhSM)
V18	21:04	How to set up a livestream for churches who want to stream their services (yt=-av6jyKma3c)
V19	31:00	Iterative setup for dual-PC streaming and troubleshooting common issues (yt=3gGpiTrkZzw)
V20	23:15	Tutorial showing how to use 20 different features of OBS Studio (yt=zXRNPozVRZg)
V21	11:26	Setting up dual-PC streaming and configuring software to support it (yt=47ZJWFHZuV0)
V22	28:59	Comparing several video switcher options for livestreaming (yt=UjFqwu3Gumo)
V23	08:22	A budget audio/video setup with tips for maximizing quality at low cost (yt=_3ZW4MAhM2w)
V24	21:27	Setting up and using OBS and its features for brand-new streamers (yt=EuSUPpoi0Vs)
V25	06:42	Livestream setup for musicians wanting to stream during the pandemic (yt=2b6_iDBU2wg)
V26	07:00	Configuring a multi-camera livestream setup using OBS (yt=8UtXvJq-l5M)
V27	10:42	Budget livestream setup for under $500 USD (yt=qMgWYem6O2U)
V28	06:43	Choosing audio and camera options for a livestream setup (yt=9EXdlHv8VXM)
V29	16:41	Tips for improving a stream on low-budget PC equipment (yt=T8gJWDaWcU8)
V30	18:05	Livestream setup tour for two sisters who stream together (yt=csYP54xZupI)
V31	14:33	High-budget streaming setup tour and showcase (yt=Tdo9iY5Lyx8)
V32	14:51	Guide for making even budget microphones sound good while streaming (yt=C6QPS3DlYKI)
V33	07:37	Creating a streaming setup with only a laptop (yt=hEUJQ4Q8SHg)
V34	06:56	Artist-focused livestreaming setup tutorial (yt=4WVHQwqxn7Q)
V35	09:11	Soundproofing choices and how some premium solutions aren't worth it (yt=VuTi4ntMA8Y)
V36	12:25	OBS tutorial and guide for setting up a USB webcam microphone (yt=DZnkyq4kqkE)
V37	06:12	Setting up a multi-camera livestream and configuring software to support it (yt=2iuk8txffrw)
V38	14:59	$1200 budget streaming setup including PC and peripherals (yt=7xggjnvT_Ok)
V39	16:32	Comparison of the top streaming software available for macOS (yt=6FIBZqFVv7I)
V40	08:42	Tutorial for setting up OBS to stream in 1080p (yt=muwqdMQptKo)

5.2.1 Data Collection Methodology

To find these kinds of livestreaming equipment videos we searched YouTube in Jan–Feb 2022 for terms such as ‘livestream setup’, ‘livestreaming guide’, ‘how to livestream’, ‘livestream software recommendations’, and ‘livestreaming setup comparisons’ to look for videos where streamers either walked through their setups or gave step-by-step tutorials about how to set up specific equipment. (Note that although many people stream on Twitch, in our experience they also have a personal YouTube channel they use to upload walkthrough or how-to videos about their technical setups.)

Each of our YouTube searches yielded dozens or more results (YouTube currently does not show the exact number since it produces additional results dynamically as users scroll down the page in an ‘infinite scroll’ UX pattern). Anecdotally we noticed that the first 10–15 results for each search were the most relevant. We used additional heuristics such as number of video views to filter, although those were often correlated with search ranking. We also eliminated videos that were advertisements or reviews for a specific product, since we wanted to find naturalistic videos of streamers describing their actual setups. Critically, we strove to sample a diverse variety of video types covering: 1) different domains of streaming (e.g., art, gaming, programming), 2) varying budget levels (from low-budget to more expensive setups), 3) those made by both popular ‘celebrity’ streamers along with less well-known streamers (e.g., an amateur musician from Singapore or someone who set up a small church livestream), 4) varying breadth of coverage (e.g., focusing on using one piece of hardware in-depth versus a general tour of the streamer’s entire room setup), 5) different genres ranging from step-by-step tutorials to more casual walkthroughs of a streamer’s home studio.

5.2.2 Data Overview and Analysis

We watched each video to perform a media content analysis, which is a standard qualitative research technique in fields such as communications and media studies [100]. This method involves treating the media itself as the primary subject of study. In particular, we noted how each streamer described their setup, what hardware and software components they mentioned, and what design tradeoffs and challenges they brought up (e.g., pros and cons, alternatives they considered using, etc). When they mentioned specific pieces of hardware or software, we went to the official product websites and looked at its technical documentation when available. Some streamers also linked to supplemental websites in their video descriptions, which we also read.

We used a combination of deductive and inductive techniques to guide our analysis. Specifically, we set out to formulate a design space[1] out of our observations of livestreaming setups. We were inspired by the methodology of Segel and Heer [125], who formulated a design space of narrative visualizations by doing media content analysis of 58 interactive news webpages, and Lau et al's design space of computational notebooks [82]. Although we started with this high-level goal, we came up with the specific dimensions of the design space via an inductive process [23] where the research team watched each video individually, met multiple times to merge our analysis notes, and categorized them together into themes. We made several iterations as a team before finalizing our 10 dimensions and representative examples within each one (see Figure 5.2). We split or merged themes as necessary to aim for a parsimonious summary, while acknowledging that there will be ambiguities present. For instance, many types of hardware may count as 'accessories' but we put some into more specific themes like stream control or lighting.

We decided to stop after 40 videos since we felt we had reached a reasonable qualitative saturation point [123]: the same themes in our design space kept re-appearing in subsequent videos we watched, and it became harder to discover uniquely new themes.

To double-check that we did not miss any major themes by relying solely on videos, we also searched Google for blog and forum posts that described streamer setups. The few that we read through covered similar sorts of information as the videos in Table 5.1, although they

[1]In HCI research, a *design space* succinctly captures multiple dimensions of variation in possible system features within a given domain [82]. See Section 2.2.4 for prior research on formulating design spaces. In our case, each livestreamer's setup covers a specific range in the design space we developed.

tended to be less detailed.

We formatted our design space diagram to match Lau et al's design space of computational notebook systems [82], with variations within each dimension plotted from left to right to represent lower to higher-fidelity, respectively. Although there is no universally agreed-upon definition of fidelity, we use the following operational definitions: A *low-fidelity* equipment option is usually simpler, easier to set up, and more novice-friendly, but it lacks customization features that professionals want. A *high-fidelity* option tends to have a higher barrier to entry and is not as novice-friendly, but it has more customization options that can be used to achieve professional-grade performance on factors such as processing speed or audio/video quality. Fidelity is correlated with monetary cost, but not in all cases: For instance, OBS Studio [1] is free and open-source software but can be configured to produce high-fidelity livestreams.

5.2.3 Study Scope and Limitations

Our design space analysis focuses on *individual desktop-based livestreaming* where a single streamer broadcasts from a static fixed location such as a room in their home or office. This is the most common setup for streaming on sites such as Twitch and YouTube, and it is what is studied by most prior research in Section 2.2.2. Note that this scope means our design space does *not* encapsulate the range of mobile and IRL (In Real Life) setups where, say, someone walks around a city streaming from their smartphone camera [92, 134] (e.g., on Instagram Live). It also means that we do not cover equipment that is meant to be operated by a production team of staff that coordinates alongside the streamer, as would be the case for more professional venues like live television shows or Esports (video gaming) tournaments in a stadium.

Our video corpus came only from YouTube, so we might be missing insights posted to venues such as blogs, forums, or other video platforms. In our experience, YouTube is now a standard place for streamers to upload their setup videos, and we strove to sample a diverse variety of such videos (Section 5.2.1). For instance, even many Twitch streamers upload clips to YouTube since it is harder to archive and search through videos on Twitch. That said, some might post relevant short-form impromptu thoughts to TikTok, so that can be an emerging domain to explore in future work.

Also, our design space is derived solely from media content analysis [100] of 40 videos, but we did not directly survey or interview streamers. We used what streamers talked about in these videos as our primary data sources, but those verbal descriptions could be incomplete. There can also be selection bias in our corpus of 40 videos due to our particular search terms. To address this limitation, in future work we could show our design space and representative examples to streamers to get their direct feedback and see what elements they would add or remove.

5.3 The Design space of Livestreaming Equipment

This section presents the results of our design space analysis, which we summarized in Figure 5.2. We report on general trends that we observed across multiple videos in Table 5.1. When relevant, we report direct quotes spoken by streamers within specific videos, labeled by video ID. For instance, the streamer in video V09 mentioned that "sometimes gear can improve the quality of your stream and can improve the quality of your life as the streamer."

5.3.1 Broadcasting

Our first group of design space dimensions relates to the live broadcasting portion of a streamer's production, which includes software that captures their computer screen, as well as all of their audio, video, and other multimedia sources. This software then organizes these into 'scenes' that they broadcast through a platform such as Twitch.

For example, in Figure 5.3 we see a streamer who is using the popular open-source tool OBS (Open Broadcaster Software) [1] to configure an artistic overlay image and his camera for an 'intermission' scene, where he can sit and interact with viewers during breaks in between streaming his gameplay content.

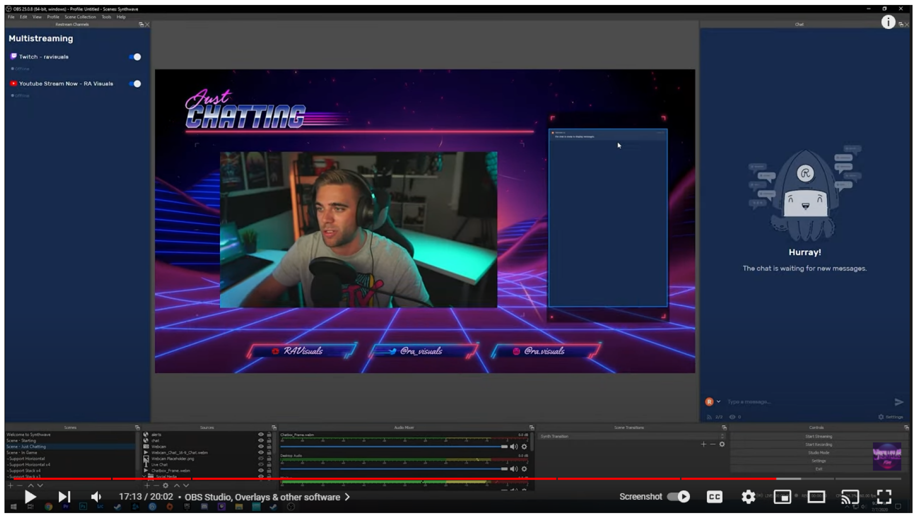
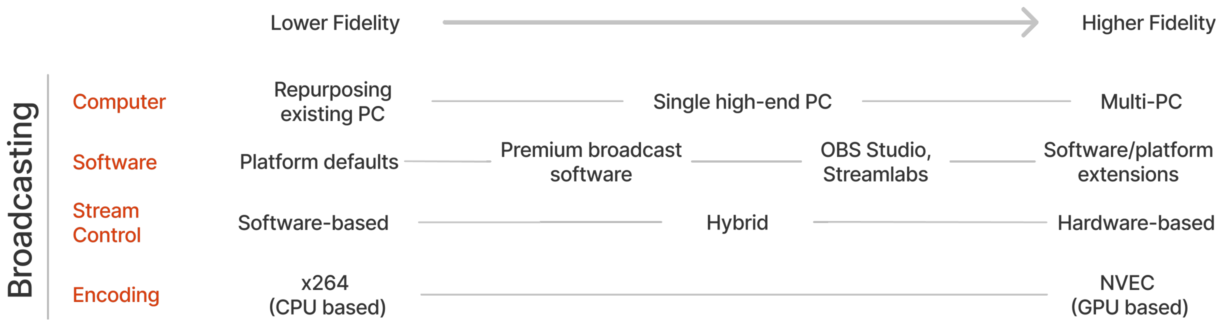

Multistreaming
Twitch - ravisuals
Youtube Stream Now - RA Visuals
Just CHATTING
RAVisuals
@ra_visuals
@ra.visuals
Hurray!
The chat is waiting for new messages.
17:13 / 20:02 • OBS Studio, Overlays & other software
Screenshot

Figure 5.3. A streamer uses the popular OBS Studio [1] broadcasting software to configure visual scenes, transitions, audio, video, and other extensions to broadcast live on Twitch.

This group has four dimensions (replicated from Figure 5.2), which we discuss in the following subsections:



For each dimension we present commonly-suggested options ordered from lowest to highest fidelity.

Computer

Streamers suggested three options for the personal computer (PC) they use to broadcast their streams:

Repurposing existing PC: The lowest-cost option is to take an existing PC and repurpose it as a streaming PC by installing broadcasting software and connecting it to all of the acquired streaming equipment like cameras and microphones. Many streamers started out here, as it is an affordable method to dip their toes into streaming. However, the resource-intensive nature of streaming can lead to performance degradation, especially for streams with graphical overlays and animations, running intensive applications like computer games, or broadcasting high-quality audio and video. Some of the tutorial videos in our analysis touched on different ways to support budget or repurposed PCs, which involve lowering the quality of stream output and using streaming software configurations to relieve the strain on a lower-end PC (V18, V23, V27, V29, V38) until one manages to upgrade to a high-end PC or multi-PC setup.

Single high-end PC: Instead of using an existing PC, which was probably not designed for streaming, many streamers recommend custom-building a high-end PC for this purpose. Due to the resource-intensive nature of both running the actual applications to stream (e.g., a modern 3-D game) and the broadcasting software along with video and audio feeds, these PCs usually require higher specifications than normal ones and can reach costs of $2000-5000+ USD.

Multi-PC: Professional streamers often use a second purpose-built high-end PC that is dedicated to handling the streaming broadcast, while their main PC runs only the applications necessary for accomplishing their intended task (e.g., creating art, playing a game, programming). Enabling streaming from a multi-PC setup requires extra hardware like capture cards to feed the main PC's live video signal into the streaming one (see Section 5.3.2), which increases the complexity and cost of the final setup. For example, depending on the desired quality (e.g., 1080p vs 4k resolution) and whether high-resolution cameras are attached, this setup may require more expensive hardware, which can raise computer-related costs to up to $10000 USD.

Software

Having chosen a computer setup, a streamer must then pick what broadcasting software they want to use. This software captures their desktop screen, running applications, audio, video, and other multimedia sources, then organizes these components into ‘scenes’ that they broadcast live to a platform such as Twitch or YouTube.

Platform defaults: Platforms like Twitch and YouTube provide their own default free software that makes it easy for newcomers to quickly start streaming. However, these are the lowest-fidelity since they lack many features found in higher-fidelity choices like OBS (e.g., support for capture cards), so streamers looking to customize their stream will not use these. Free defaults are also tied to each platform, so switching platforms or multi-streaming (streaming on multiple platforms at a time) is not possible. As a result, streamers in our videos rarely mentioned this option.

Premium broadcast software: Several streamers (V13, V18, V31, V39) talked about premium broadcast software like Ecamm Live [29] and VMix [90] that, for some cost ($15-50 USD per month or $50-1200 for a single license [33]), provide features that are not available on the free platform defaults. For instance, V39 mentioned:

> *“They also make it so easy just to move and adjust stuff on the fly while you’re live. So if you wanna zoom in on a screenshare so that your viewers can really see what it is you’re showing them, you literally just pinch to zoom on your trackpad and it’s gonna zoom in on that screenshare.”*

Some also include filters and effects that change the live video and audio, and multi-casting to different platforms.

OBS Studio, Streamlabs: By far the most widely-used option amongst streamers in our analyzed videos was OBS (Open Broadcaster Software) Studio [1] and software that extends OBS, such as Streamlabs [122]. OBS is open-source and free, but forks like Streamlabs do have subscription fees ($20 per month) that provide other features like overlays and multi-streaming support. Despite being free[2], OBS Studio can result in higher-fidelity streams than the above category of premium software since it is designed as a power tool for advanced users. However, the downside of OBS’s flexibility is that it is harder to set up. For instance, V39 mentioned how

[2]Higher-fidelity usually means higher monetary cost, but for software there are free options that can result in higher-fidelity outputs than paid ones.

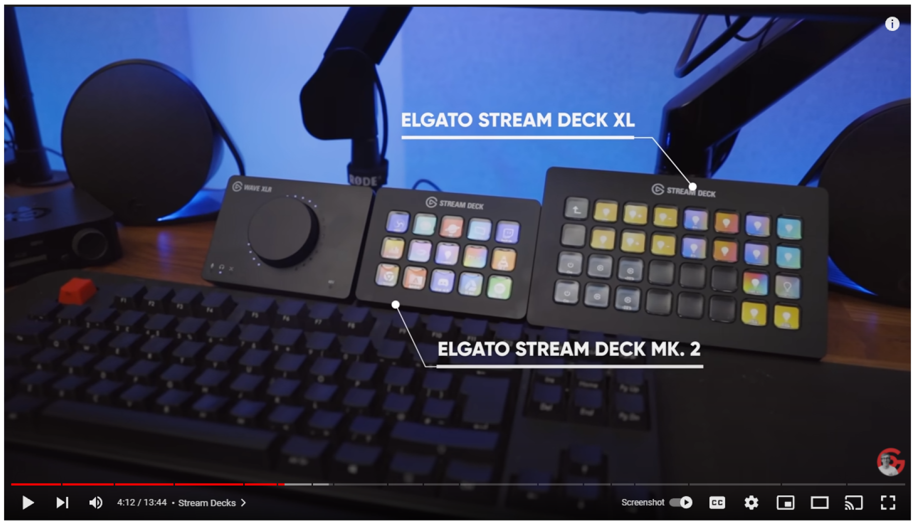


Figure 5.4. The Elgato Stream Deck [32] is an example of a hybrid stream control interface. It contains programmable buttons (with customizable mini-LED displays on them) to control streaming software like OBS using macros that the streamer specifies.

OBS has a steep learning curve: "Simple things like not having any templates, or presets for different livestream qualities, or the different platforms just means that you actually need to know what you're doing or go and find out what the correct settings are to get the best results."

Software/platform extensions: For even greater fidelity, streamers can install extensions and plug-ins to their broadcasting software to customize stream appearance and functionality (e.g., adding viewer donation alerts). Some extensions are offered by streaming platforms like Twitch [139, 140] (e.g., a Twitch extension for tracking the streamer's eyes [35]), while others control broadcasting software remotely (e.g., OBS-websocket [111] can switch OBS scenes remotely or have viewer donations automatically trigger visual/audio effects).

Stream Control

While they are broadcasting live, streamers need ways to control the appearance of their stream and adjust it in real-time. Broadcasting software (see prior section) come with some built-in controls, but advanced streamers prefer dedicated hardware to control their streams rather than relying solely on their mouse and keyboard.

Software-based: Software like OBS lets streamers activate different scenes on-the-fly, like an 'intermission scene' that has their facecam (main camera pointed at their face) made to fill up most of the screen or a 'gaming scene' where the facecam is made smaller to fit in the corner overlaid onto the game being played full-screen. This is the least convenient option, though, since it requires the streamer to move away from what they are focused on doing on-screen (e.g., playing a game) to navigate separate menus in the broadcasting software or to use a separate set of keyboard shortcuts.

Hybrid: This approach involves using hardware input devices to control the broadcasting software (i.e., a hybrid of hardware and software). Such input devices include foot pedals ($15-90 USD) and the popular Elgato Stream Deck. The Stream Deck ($90-250 USD), shown in Figure 5.4, contains hardware buttons that streamers can bind to custom macros. Once set up, when they press these buttons it performs various stream functions like changing scenes, muting microphone or music, skipping donation messages, and anything else that broadcasting software and extensions afford. Each individual LED button on the Stream Deck can also present a customized image and text so the streamer knows what each button does. V03 said that it "feels like I have my own mission control, which adds sort of a fun dimension to streaming." Streamers have found these types of devices to be incredibly useful as "pretty much every task that I'm doing day-to-day as a content creator has some functionality control through one of these stream decks." (V01)

Hardware-based: On the highest end of fidelity and price, there are several purely hardware-based solutions for stream control called *video switchers* (Figure 5.5). These can provide scenes, transitions, features like picture-in-picture, auto-camera selection, audio mixing/levels, and more in a single hardware solution without needing to interface with broadcasting software on the streamer's PC. Thus, these devices are seen as more reliable than software solutions, especially

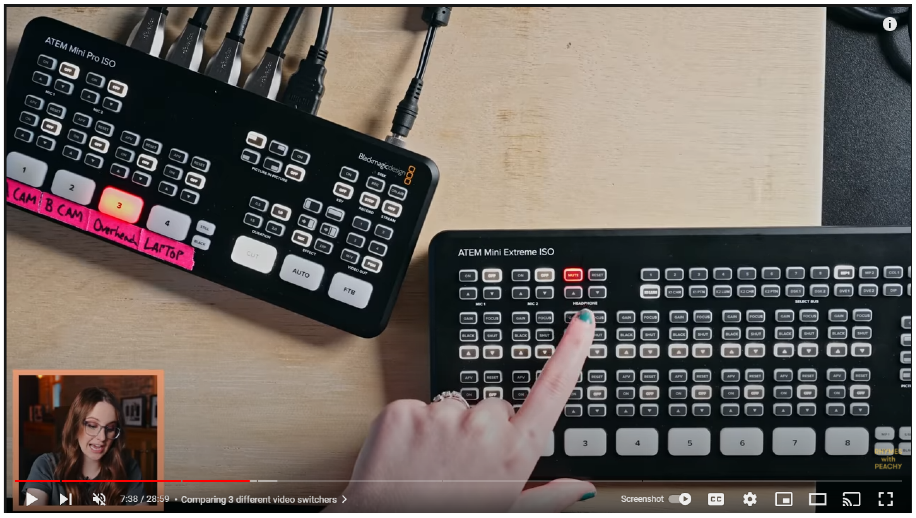


Figure 5.5. This streamer uses a hardware video switcher to customize scene composition (e.g., picture-in-picture) and transition between scenes in her stream. Doing so in hardware takes the processing load off her computer and can thus be faster and more reliable.

for more complex setups that involve multiple video and audio inputs. These are more likely to be used for larger productions such as V17 and V18 (e.g., conferences, podcasting, or church streams), come with a much higher price tag ($600-5000 USD), and thus are less accessible than hybrid solutions like the Stream Deck.

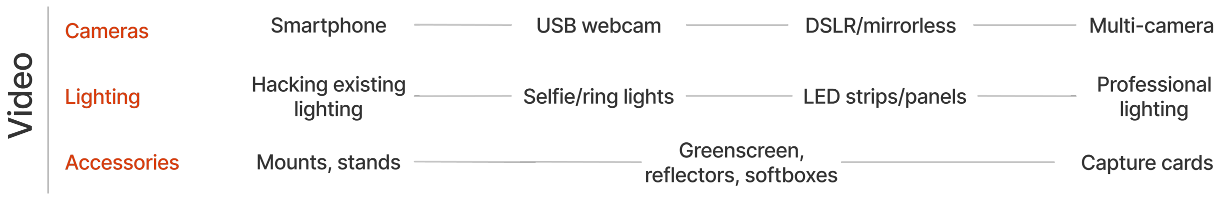
Encoding

Finally, streamers weighed the pros and cons of the two main video encoding methods for livestreams. The lower-fidelity option is x264 CPU-based encoding, which is for lower-budget streams. However, the preferred solution was NVEC GPU-based encoding, which requires an expensive Nvidia video card ($250-2500+) but can handle higher quality streaming. Some also mentioned that due to recent GPU chip supply shortages [136], it may be harder for people to obtain NVEC-capable hardware for streaming, so x264 with a higher-end PC may also be adequate.

5.3.2 Video

The next group of design space dimensions relates to hardware to provide a live video feed of the streamer and their physical environment (e.g., their room), which can increase personal engagement with viewers above and beyond providing just a screenshare of their computer (e.g., showing a live game or image editing program). V07 said that while it is possible to stream without a camera (and several popular streamers do), “In my personal opinion it really disconnects the viewing experience between you and your viewer if you don’t show your face on stream.”

Here are the three video-related dimensions that streamers mentioned:



Cameras

Streamers must first choose a camera to broadcast their face to show live reactions, real-space physical work (e.g., drawing, painting, playing musical instruments), or to interact with viewers.

Smartphone: While most streamers recommend at least a USB webcam, some showed how to repurpose an existing smartphone into a camera that rivals most webcams. But wear-and-tear on the device might make this a temporary solution until one can invest in a dedicated camera. The setup of a smartphone as a streaming camera also requires installing a mobile app to broadcast video output onto the web that streaming software can add as a source, and it may also become tedious to constantly mount and unmount it from a fixed stand near the computer.

USB webcam: These are easy to set up since they connect to the computer without extra hardware that a DSLR/mirrorless camera needs (see below for details). Webcams usually come with a built-in microphone, but most of the streamers who used webcams do not recommend

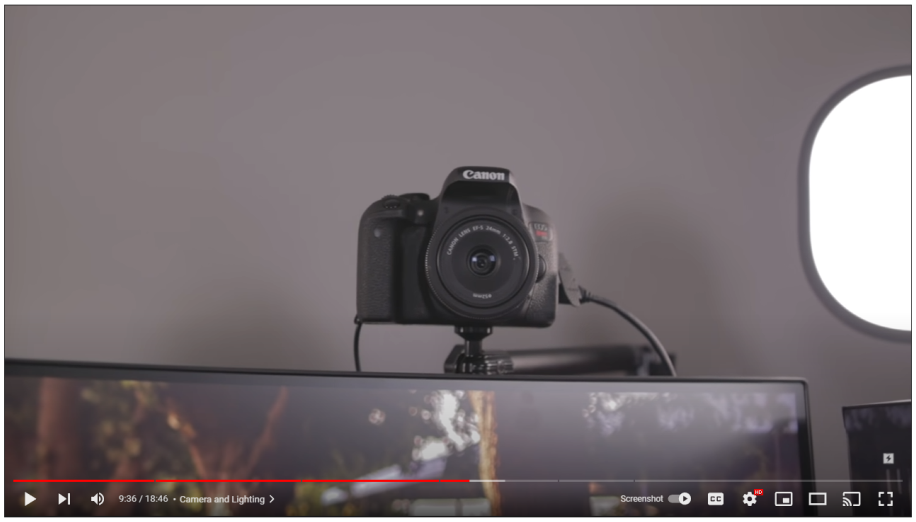


Figure 5.6. A DSLR camera configured to be used as a high-fidelity facecam (face camera), mounted above the streamer's main monitor.

relying on it (see Section 5.3.3 for microphone choices). V13 also noted that high-intensity lighting can also wash out the image of most webcams, so care must be taken when using one.

DSLR/mirrorless: As a high-fidelity option, streamers recommend DSLR or mirrorless cameras used by photographers (Figure 5.6). These provide advanced features like sharp autofocus when the streamer moves around, depth-of-field to blur their background, and changing the ISO (photo sensor sensitivity) or white-balance. However, these cameras can be expensive for streamers just starting out ($1000-$3000+ USD), so the general recommendation is to get a webcam first and then upgrade later. Also, these cameras must either be connected to a hardware video interface (see Section 5.3.1) or to a PC via a capture card; both options increase cost and setup complexity.

Multi-camera: The highest-fidelity option, a multi-camera setup, gives viewers multiple views into the streamer's physical environment (see Figure 5.7). While one camera may be the main facecam, other cameras can provide overhead shots of a drawing tablet, musical instrument, or other device the streamer is using, as well as wide shots or other views the streamer might want

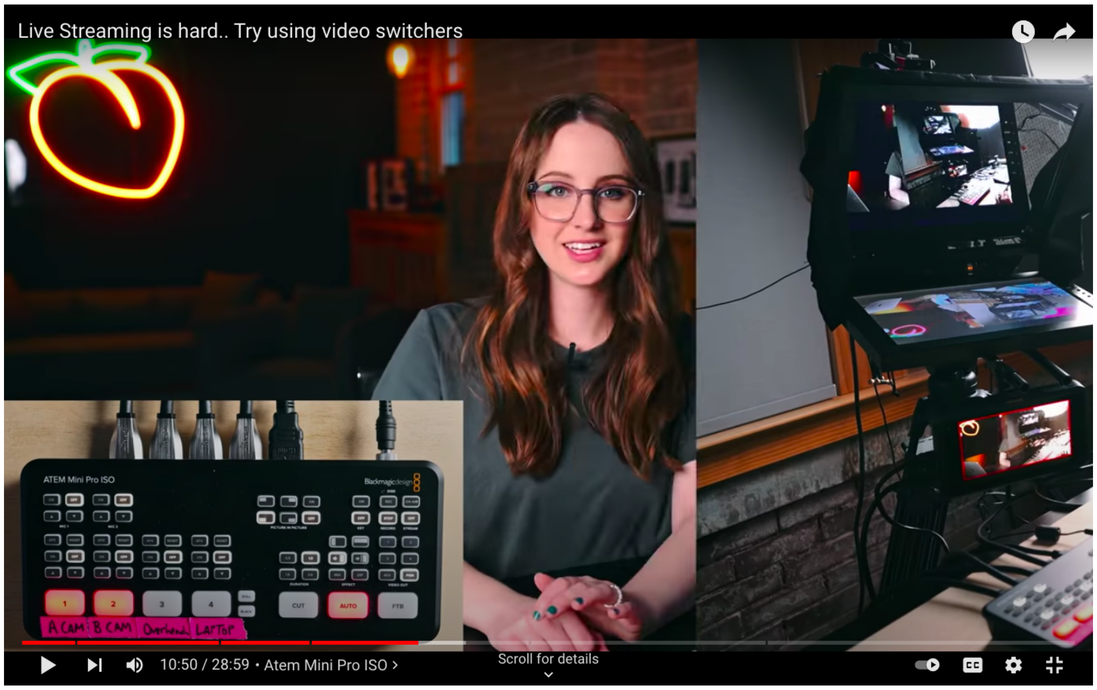

Live Streaming is hard.. Try using video switchers
ATEM Mini Pro ISO
A CAM B CAM Overhead LAPTOP
10:50 / 28:59 • Atem Mini Pro ISO >
Scroll for details

Figure 5.7. A tech review streamer uses a multi-camera setup controlled by a hardware video switcher to broadcast 3 camera angles.

to share. Using a multi-cam setup introduces complexity in terms of stream control, which may lead to needing a video switcher (Section 5.3.1) or several capture cards and a powerful enough computer to manage all the video sources. V13 mentioned the importance of coupling high-end cameras with dedicated video switcher hardware: "Having a physical hardware solution can make it a little easier so it's not absolutely critical but in my case it simplifies matters greatly because now I can have up to four different cameras with a single device."

Lighting

Streamers mentioned that lighting is "the secret to a good-looking stream" (V04) but novices may overlook it. For example, novices who use only their monitor to light their face can appear jarring to viewers as dark scenes in games will make it hard to see their face and a sudden brightness change on-screen can overexpose the video.

Hacking existing lighting: Lighting can get expensive, so some budget-minded streamers gave advice on how to repurpose (hack) existing lighting in one's home to mimic a higher-end setup

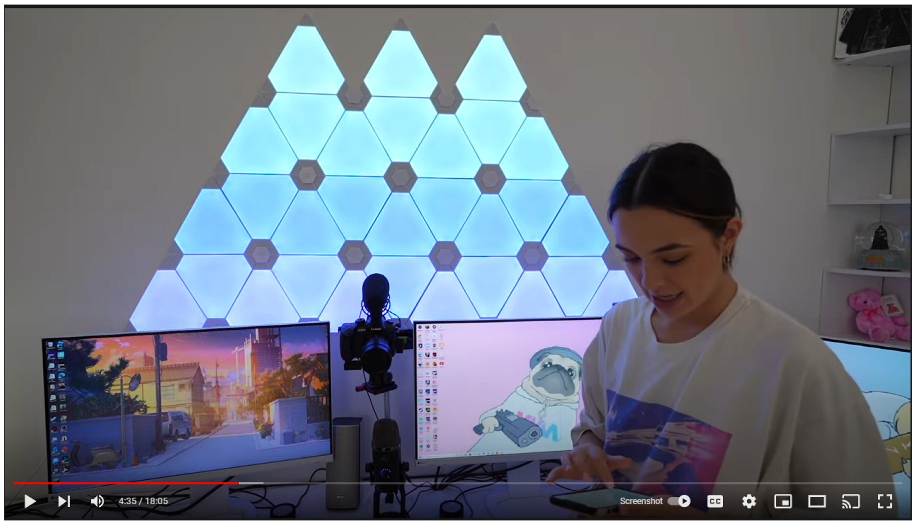


Figure 5.8. Example of using multiple wall-mounted LED panel lights as a key light (main source of light) to illuminate the streamer while she is on camera. These lights can be programmatically controlled via a mobile app.

for almost no cost. For example, instead of purchasing light boxes, which can be expensive and require a large amount of space to rig up, use an old lamp with a halogen bulb, which can get consistent and true-to-life colors out of the camera. Then add partially-transparent paper (e.g., wax paper) to diffuse the light, or even just bounce the light off the wall to soften its glow.

Selfie/ring lights: Some streamers aiming for budget setups used selfie lights (ring-shaped LED lights that a camera can sit in the middle of) as their *key light* (main source of light). These are relatively cheap ($20-100 USD) compared to professional lighting rigs while still providing decent light for the streamer to show their face on-screen.

LED strips/panels: These are generally used for background, fill, and ambient lighting and allow the streamer to add color and personality to their stream. For instance, V13 mentioned how LED panels add a "little splash of color up on the background to create some depth and separation of me from the background." Several also showed how to use LED panel lights as their main key light, which can save money over buying professional key lights (see Figure 5.8).

Professional lighting: Some streamers recommended expensive lighting setups to make videos look professionally-produced. These setups are similar to what photographers use and can run over $1000+ USD.

Accessories

Finally, accessories can increase production value by enhancing the streamer's video. Since there is a wide variety of accessories that streamers recommended, we sorted them by monetary cost as a proxy for fidelity:

Mounts, stands: Mounts and stands provide stability and reduce vibrations for cameras. They also let streamers install lighting and cameras at the angle and height most conducive for good-quality video. These are cheap compared to the rest of the setup (starting around $15 USD), though some streamers suggested free alternatives like bending a metal clothes hanger to lean their smartphone camera against as a makeshift mount.

Greenscreen, reflectors, softboxes: Portable greenscreens can be used to hide the streamer's background environment for privacy. They also allow the streamer to display a digital background, which can add a personal touch without investing the money to build an elaborate physical set or stage behind them. Reflectors help redirect light which can help "light up both sides" of a streamer's face (V01), and softboxes soften the light coming from their lighting setup.

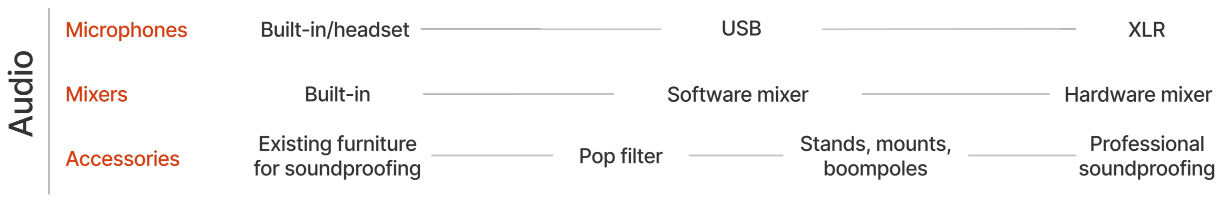
Capture cards: Camera capture cards are important for connecting DSLR and mirrorless cameras to the streaming PC. USB webcams do not need this extra hardware as they can interface directly with a PC, but streamers who use high-end camera setups or multi-PC setups need a capture card like an Elgato Cam Link [31] in order to complete their setup. Console capture cards are used in a similar way to capture footage from video game consoles.

5.3.3 Audio

Audio hardware allows streamers to increase engagement by speaking to viewers and also manages audio levels for software (e.g., a game) or other content (e.g., a YouTube video clip) that the streamer is showing. Audio is seen by some streamers as the most critical upgrade that novices can make to improve quality. For instance, V32 said:

> *"In the film industry we have a saying that 60 percent of your film is audio. But when it comes to streaming on Twitch or generally streaming a lot of the time, it's actually 100 percent because people put you on in the background or they play a game or they do something else. So having good audio is absolutely crucial."*

Streamers mentioned three audio-related dimensions:



Microphones

These broadcast the streamer's voice and external sounds like musical instruments they are playing.

Built-in/headset: Microphones that come built into webcams or headsets are the lowest-fidelity choice and not recommended by experienced streamers. That said, budget-conscious streamers find that a gaming headset is a good starting option when nothing else is available, and many streamers mentioned how they originally started with one.

USB: USB microphones are perceived as medium-fidelity and can be found at affordable prices ($30-150 USD). Since USB mics have integrated amplifiers and DACs (digital-to-analog converters), they do not need additional audio hardware, which makes them easier to get started with. However, some streamers mentioned they had issues with the integrated mixers that come with USB microphones and that the quality of the integrated amplifiers and DACs can be sub-par.

XLR: Figure 5.9 shows an XLR microphone, which is a professional-level option for streamers

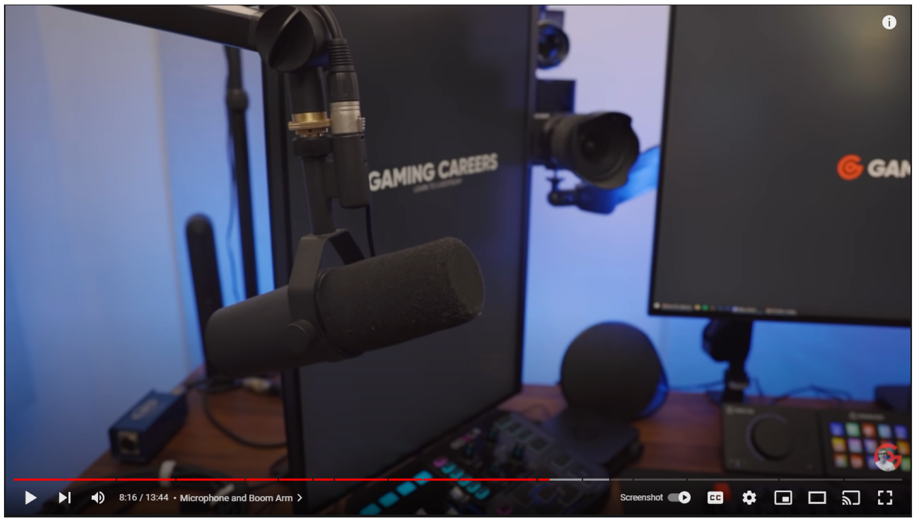


Figure 5.9. A high-fidelity XLR microphone on an adjustable boom stand (with two DSLR cameras in the background near the monitor).

who want to get the best sound "because they send a balanced signal that isolates noise" [146]. These capture an analog signal to send to an audio interface (see Section 5.3.3) that converts it to digital to send to the computer. XLR microphones can get expensive ($400+ USD), but there are lower-end ones that can open up more options down the road because streamers can upgrade different parts of their audio setup (e.g., buying a better audio interface) without having to buy a new microphone.

Mixers

This dimension encompasses the range of hardware and software interfaces needed to mix and manage audio inputs/outputs. Audio mixers are an important part of a streamer's setup as it allows them to adjust audio levels like sensitivity, gain, and volume to react to whatever is happening live during a broadcast. For example, V06 stated that desktop audio should be set at 50-60 percent while the streamer's microphone audio should be at 100 percent in order not to drown out the streamer when the applications they are streaming get too loud.

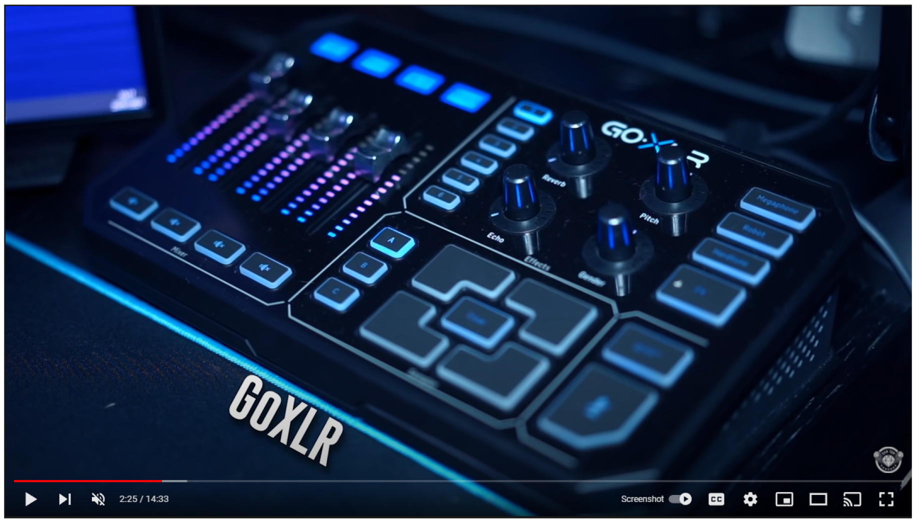


Figure 5.10. A hardware audio mixer that supports connections with XLR microphones and other digital audio sources. It provides buttons for muting individual sources, applying sound filters and voice effects to each, and physical sliders for fine-tuning audio levels.

Built-in: The lowest-fidelity audio interface is simply using the built-in audio mixing capabilities of broadcasting software (e.g., OBS) or the operating system (e.g., Windows audio level controls). This is a simple and free solution for those who do not want to adjust or configure the complex controls that more advanced software or hardware provide.

Software mixer: For those using USB microphones or who desire finer-grained sound control, streamers pointed out several software mixer options. For example, Voicemeeter Potato [9] is an audio mixer that can manage any audio device connected to the streamer's PC. However, some streamers had reservations about software solutions for audio mixing as it can require a lot of configuration and may be taxing on the CPU. For example, V03 drew up a diagram that she uses when she needs to debug any audio issues "and it feels like my brain needs to do back-flips to understand it all."

Hardware mixer: The highest-fidelity option is a dedicated hardware audio mixer, which is required for connecting high-end XLR microphones (Figure 5.10). These interfaces can also come with hardware adjustments for sound levels so that a streamer can adjust their sound levels without operating additional software.

Accessories

Similar to video accessories in Section 5.3.2, here we sort recommended audio accessories by cost:

Existing furniture for soundproofing: Using curtains, drapes, and rugs to reduce echo in the room the streamer is in can improve sound quality when streaming from locations with poor acoustics, such as rooms with concrete floors or large windows. These options are usually free or low-cost since they use the streamer's existing household furnishings.

Pop filter: This is a $10-20 USD cover that protects the microphone's condenser to prevent unpleasant popping noises when streamers speak with B or P sounds. It is frequently recommended as a low-cost way to improve audio quality.

Stands, mounts, boompoles: These accessories can prevent noises that occur when streamers hit their keyboard or desk too hard. An adjustable boom arm is preferred since it saves space and allows the microphone to freely move.

Professional soundproofing: While using existing furniture to lower ambient sound can help and "is one of the key ways to cut down on audio reverb" (V11), many streamers invest in professional soundproofing such as modular panels of foam or other material installed on walls or ceilings. Some believe this to be a worthy investment ($100s-1000s USD) as "it is hard to attract viewers if you sound like you are streaming at the end of a sewer pipe" (V05).

5.4 Discussion: Trends, Challenges, and Opportunities

We now zoom out from individual design space components to pose four broader questions inspired by our analysis:

1. How can we help livestreamers integrate many separate components into a unified setup?
2. How can we help streamers manage the complexity of their equipment while broadcasting live?
3. How can we teach novices to incrementally upgrade their setups without getting overwhelmed?
4. How can livestreaming setups inspire future tools for remote work and education?

5.4.1 How can we help livestreamers integrate many separate components into a unified setup?

While the acquisition of higher-end equipment is one barrier to increasing stream fidelity, streamer frustrations also stemmed from the complexity of integrating all of the hardware and software together and making the correct configurations to properly use that equipment with one another. When streamers run into issues and need to debug their systems, it can be difficult to navigate the complexity of a high-fidelity stream setup, especially because not every streamer will have the technical expertise to dive into configuring each piece of technology in it. For example, V19 said the following about upgrading to a dual-PC setup:

> *"I was so stressed! If you're also setting up a dual-PC setup, don't worry. I understand your stress. Been there, done that! I get it. This video is going to consist of all of the technical difficulties that I ran into while setting up my new setup. There's plenty of tutorials online that can say 'hey here's exactly how to do this,' but they never tell you what happens if you run into a brick wall."*

While the streamers in our videos attempted to thoroughly explain their setups, they often left useful knowledge unsaid. With the vast amount of equipment to individually configure, plus further integration of this equipment with broadcasting software, streamers might not remember to share all the details of the options that they used and the 'insider knowledge' required to debug their setups. So, how can we help streamers share this knowledge with each other to enable novices to get past the frustrations involved in integrating their equipment together?

To address this challenge, we envision a novel tool that can elicit knowledge from streamers, collect data on both the physical and digital aspects of a stream setup, and share it online. This tool can identify knowledge gaps in the integration of components to guide streamers who are using similar components. Whenever a relevant component combination is detected within a streamer's current setup, a set of verified physical and software configurations can be presented to them. This tool could also provide a way for streamers to create technical blogs and vlogs where they share frustrations and successes while integrating their equipment into a complete setup. These blogs/vlogs can be synthesized, organized, and made searchable so that novices using similar components can quickly find example walkthroughs that are more targeted to their own goals. This tool might also detect issues with equipment and integration between parts of the setup that the streamer has acquired and present contextually-relevant fixes found online.

5.4.2 How can we help streamers manage the complexity of their equipment while broadcasting live?

The challenge doesn't stop once the initial setup is done. Running these streaming setups live while playing games, musical instruments, creating art, etc. is difficult since the streamer is often alone. Online moderators can sometimes help with operating web extensions that allow them to control scene selection and other interactions, but since *the streamer is both the director and performer* they need to be able to make changes to their stream themselves on-the-fly. For example, a streamer might need to adjust multiple audio controls for the same microphone, muting themselves in the in-game voice chat so that they can talk with viewers while playing a game without distracting the other players they are with, or mute themselves from viewers when they need to speak to someone privately. Artists and chefs may need to swap between multiple camera angles as they work and move around in their studios or kitchens. And variety streamers [45] may need to change their stream scene layout with each new game or app they open so as not to accidentally obscure important UI elements that are inconsistently placed between each application. Given these challenges, how can we support these streamers into becoming their own single-person live production crew?

To this end we envision a novel tool that can automate critical parts of livestream production. For instance, when this tool detects potential stream overlay or layout issues, it can warn the streamer that their existing scene may need to be modified to prevent hiding parts of an application's UI that might be interesting for viewers (e.g., the health bar of their game character). When a new application is launched, the tool can provide recommendations for where the facecam and other parts of their stream overlay should be positioned, informed by a heatmap of previous streamer layouts within these specific apps. To control scene selection, this tool might detect movement on other cameras in a multi-camera setup and switch to a scene that shows the most active camera to viewers. However, such a feature would need to allow the streamer to define when this automation was allowed or prohibited (e.g., during a game the streamer would likely not want to auto-switch to a wide-angle shot of their room). Further, this tool might detect what applications are being used and select scenes based on what application is in the foreground of the streamer's PC and the current state of that app. For example, when a programming streamer launches their IDE right after talking with their viewers for a bit, this tool could control OBS Studio to change from the 'just chatting' scene to a 'programming' scene that shows only the IDE. If they encounter a code error while debugging, perhaps the tool switches to an 'error' scene that zooms into the stack-trace within the command-line and plays a sad violin music clip for the audience.

5.4.3 How can we teach novices to incrementally upgrade their setups without getting overwhelmed?

Livestreaming equipment can be difficult to acquire and set up all at once. It can be hard to balance one's targeted stream fidelity with the monetary investment one has to make to achieve it. Some streamers noted that everyone needs to find a balance that works for them. Ironically, watching streaming setup walkthrough videos may make novices feel intimidated by everything on display. For example, V16 shared their frustration with streaming setup videos "because I find these videos somewhat pretentious from the get-go. This whole thing [my streaming setup] is 10 years' worth of collecting stuff. It looks more impressive and expensive than it actually is, and some of the stuff I even got for free [from sponsors]." So how can streamers incrementally upgrade their setup without getting overwhelmed?

V13 recommends starting one piece at a time "because livestreams are hard. It is a lot going on, especially if you're directing it at the same time that you're in it and featured in it. So, for me I would actually try to work on all of this technology one piece at a time." But even with this advice, novices may need more detailed guides of how to incrementally ramp up without getting overwhelmed. For example, a novice streamer may begin their streaming journey with just a USB microphone, no external hardware audio or video interfaces, a budget webcam, and a single laptop running OBS Studio to broadcast. From here there are many paths that they could go down for smaller upgrades to improve their stream's production quality. They might first upgrade their USB microphone to an XLR one, but with this upgrade they must also find an audio interface that allows them to connect their higher-fidelity microphone. They might also upgrade their webcam to a DSLR camera, which will require further investment into accessories like a capture card. Or they might switch from software-based broadcasting to a hardware video switcher, which may itself also take over audio interfacing duties. There are many possible paths for a streamer to go down when making upgrades, which could result in choice overload.

To address this challenge, we envision a tool that recommends iterative upgrades to novice streamers given their current starting point and budget constraints. But how would these novices know what to upgrade first? While viewers of the stream may be able to give feedback to the streamer on what generally needs improvements (e.g., camera quality, audio noise, etc.), we believe this tool should also detect 'low-fidelity' stream symptoms and predict the likely causes behind them. For instance, video pixelation might lead to a recommendation to switch to NVEC encoding, or a noisy audio signal might be improved with a better microphone and audio interface setup. Viewers might still be leveraged through polls and chat feedback presented to the streamer so they can visualize areas of improvement for their own broadcast. Another alternative is to present novices with upgrade paths that previous streamers with similar setups went down, including the alternatives and low-cost repurposed ‘hacks’ they used. It can support streamers by exposing various existing setups and configurations, so that a streamer can pick and choose different elements of production fidelity that they want to reflect in their own streams. Finally, it can present before-and-after comparisons of specific components – i.e., how did this particular upgrade affect other streamers who tried it? This knowledge will allow streamers to make more informed decisions on whether the upgrade is worth it for the production quality they desire.

5.4.4 How can livestreaming setups inspire future tools for remote-work and education?

As hybrid and remote work arrangements grow more widespread in light of the current global pandemic, more people are thinking about improving their home office setups. Since a major form of communication between coworkers is via online video conferencing platforms like Zoom or Microsoft Teams, many of us are now thinking about improving the production quality of how we present ourselves in this format. Using a low-fidelity setup (e.g., a laptop’s built-in webcam and microphone) during online meetings for extended spans of time can lead to ‘Zoom fatigue’ [34, 119, 144].

Studying the equipment setups of professional livestreamers may inspire us to improve our ‘personal amateur livestreams’ that we broadcast as remote workers and students, which could in turn improve long-term remote work and learning arrangements. Specifically, improvements to video quality and livestream-inspired interaction techniques might form greater connections between team members or classmates who cannot meet in-person.

We optimistically believe that livestreamers can be trend-setters here, and we envision the home office of the future to look more like the high-fidelity streaming setups of the present. While not all approaches to high-fidelity streaming are appropriate for the virtual workplace or lecture hall, having increased control over virtual self-presentation can potentially benefit both workers and students. However, more research needs to be done to understand whether applying these higher-fidelity setups within the context of a business or university actually brings real benefits, or whether more widely-accessible low-budget setups are sufficient. What is the balance of cost and fidelity required to be effective here?

5.5 Conclusion: The Design Space of Livestreaming Setups

In this chapter we presented a content analysis of 40 livestreaming setup videos and distilled their features into a ten-dimensional design space consisting of broadcasting, video, and audio equipment. This design space captures the contemporary trends in streaming setups, along with challenges and design opportunities for improving the integration of components and helping novices get started. More broadly, we propose that the technical knowledge that livestreamers possess can potentially help a much larger population of people around the world who are now working and learning remotely. In the coming years, we predict that more of our remote and hybrid working setups may resemble what we see professional livestreamers using today.

5.6 Summary and Acknowledgements

This chapter of the dissertation investigates the current state of livestreaming setups, which can become complex depending on the individual needs of a streamer (including budget and fidelity). It presents a survey of 40 videos published by streamers that describe various aspects of streaming setups, including the setups they themselves use, potential low- and high-fidelity setups, and the challenges involved in installing, running, and maintaining them. This analysis of these videos is leveraged to categorize current livestreaming equipment and presents the design space of livestreaming setups which has ten dimensions. It presents a discussion how this design space informs future streaming support tools to improve novice and expert streamers when working with their own streaming setups. Finally, the chapter explores how these findings might apply more broadly to improving live collaboration tools in a post-pandemic world where these tools for remote work and collaboration have become critical for companies.

Chapter 5, in full, is a reprint of the material as it will appear in the proceedings of the 2022 Conference on Designing Interactive Systems. Ian Drosos and Philip Guo. 2022. The dissertation author was the primary investigator and author of this material.

Chapter 6

Conclusion

This chapter gives a summary of the contributions of the research found within this dissertation and discusses the implications of their findings and future directions for research to improve the design of novel systems for enabling complex workflows like data science and content creation. My conclusion is that these tools need to reveal the gory behind-the-scenes details found within complex workflows to enable learners in handling them so that they can become experts, instead of hiding them for a more "user-friendly" design. This dissertation presented 3 chapters which lend evidence to this thesis in the areas of data wrangling tools (Chapter 3), instructional workflow streams as a virtual form of cognitive apprenticeship (Chapter 4), and the management of complex streaming setups (Chapter 5).

Much effort has gone into assisting technical practitioners with the complex workflows these practitioners are responsible for. Systems that leverage intelligent technologies like program synthesis, artificial intelligence, and machine learning have removed barriers to performing these workflows and the sources of frustration found within them. However, we now know that there can be great value in a lot of the work that gets automated and obscured for the users of these systems. These technical practitioners are also adding to this obfuscation of potentially useful and educational information as well when trying to teach their craft to others through video tutorials or streaming their workflows live to an audience. And so, there is likely a balance to be reached that leverages the power and convenience of intelligent systems that assist practitioners performing complex technical workflows, but still keep the behind-the-scenes details that can be leveraged by these practitioners, or even learners interested in these practitioners' workflows.

I believe that if we hide away too much of these details, it can result in users having to put a lot of trust in these systems in the hope that they are doing what they say they are doing, and doing it in a way that is preferable (efficiently, effectively, ethically, etc.) for the user. However if we show these details, users can verify these tools are doing what they think they are doing and extend their solutions. This also extends to the teaching of these workflows to audiences through instructional workflow streams and livestreaming. This kind of work requires difficult and tedious steps sometimes, and hiding these frustrations from novices who want to learn how to become experts at these workflows is doing them a disservice. Allowing learners to view and explore the nuances of complex workflows can provide context to the problems and approaches to solving those problems found within them. Thus, my thesis is that *being able to see and interact with technical workflows behind the scenes can help novices grow into experts, but existing user-friendly computational systems may hinder growth by obscuring important kinds of complexity.*

6.1 Contributions

In order to support this thesis, this dissertation provides several contributions within, which are summarized below:

In Chapter 3 – a formative study was performed with professional data scientists to define the design principles of a novel data wrangling tool called Wrex, implemented as a Jupyter notebook extension, that can synthesize readable Python code that performs data transformations defined by a user through input-output examples within an interactive dataframe grid. Through a usability study with Wrex and professional data scientists performing common data wrangling tasks we found that when data scientists used Wrex they were more effective and efficient at solving data wrangling tasks than when manually programming to solve the task. Participants found using Wrex to be intuitive and noted it fit within their existing data analysis workflows and that the readable code it generated was a critical feature for acceptability of the code synthesized by data wrangling tools.

In Chapter 4 – 20 instructional workflow streams in the domains of data science, web programming, digital art, and gaming were analyzed for pedagogically relevant content that normally gets cut out of more polished videos and tutorials. The nature of these streams provided evidence that they can enable a virtual form of cognitive apprenticeship by revealing several types of serendipitous teachable moments like improvised problem solving, example creation, tangents, and high-level advice. Yet several missed teachable moments were identified, possibly due to the difficulty of conveying tacit knowledge. Finally, novel tool designs were proposed to enable content creators in creating these instructional workflow streams for sharing their own workflows.

In Chapter 5 – 40 streaming setup videos were analyzed that showed many aspects of streaming setups, including their own setups and how they configured them, advice for low- and high-fidelity setups based on budget and need, and the challenges they face when managing these complex systems in their daily work. Through this analysis, a 10-dimension design space of streaming setups is presented which encompasses the equipment used within these setups. Finally, ideas for future streaming support tools are shared to enable streamers to navigate this design space themselves for their own streaming setups and note how these findings can expand beyond content creation and inform workers participating in remote work and collaboration in industry and beyond.

6.2 Implications for future systems and their users

The work in this dissertation has several implications on the design of future systems to better serve technical practitioners and for those that are also content creators.

6.2.1 Wrangling data wrangling

In Chapter 3, we showcased Wrex, which provides readable code as an artifact along with the original goal of providing a wrangled data frame for the user. Several participants found that this readable code had great potential in helping them learn transforms, functions, libraries, and even help with learning a new language that they might be uninformed on. I see these tools as a pair-programmer coworking alongside the data scientist. Therefore we need to create systems that don't just do data wrangling, but instead teach data wrangling alongside the features that enable practitioners to more effectively or efficiently accomplish their technical workflows.

Remember Dan the data scientist from Section 1.2? Well now with Wrex, Dan's situation looks a bit better than when we started off. Now when Dan sits down to wrangle a new data frame to perform a data analysis, Dan has the option to write code as usual when he knows what code he needs to write, however if he ever gets lost or finds the code he needs to write tedious, he can interact with his data frame to produce useful source code that he can incorporate immediately within his computational notebook. When Dan wants to share his analysis, his co-workers can now inspect the code that performs the various tasks within the analysis workflow to understand and verify what Dan has done and enable the code review process ubiquitously found in software engineering. More importantly though, is that Dan's co-workers can now incorporate the functions synthesized by Wrex into their own work, even if they haven't interacted with Wrex.

Readable synthesized code might inspire a data scientist to explore new APIs and programming techniques, which can expand their own personal list of technical skills. Further, this increases the usability and reusability of the synthesized solution. It allows these data scientists to extend the code, or even work to create more performant versions of the code now that they have a readable function that they can understand that performs the tasks they desire. Readable synthesized code can also enable collaboration with other members of their team, so they can verify, reuse, or extend the code synthesized in the analysis to their own analyses. If this the readable code generated by Wrex is not available, collaborators would have to re-interact with the data frame, mimicking the actions of their co-worker, to get the same results, but this would introduce tedium that reusing synthesized wrangling code would not.

These tools and systems will, on a long enough timeline, fail. In these cases, if working with the tool is more of a learning partnership than a transaction of ‘here is my data, now wrangle it’, then the human partner isn’t just lost without the tool, as they’ve gained skills and learned new methods for working within their workflow from the tool and can handle the cases where the tool fails.

Finally, beyond computational notebooks, there is the opportunity to provide source code in tools that normally lack the ability to leverage powerful general programming languages, like Python, within the workflows created by interacting with these tools. For example, a novice business or data analyst might start with using an Excel spreadsheet to perform their data analysis with intelligent wrangling tools like Wrex, but as they do so inspect the code it generates to perform these tasks. It is the hope that they might learn from this and increase their programming skills so they might ’upgrade’ to more code-heavy workflows once their experience grows.

6.2.2 Instructional workflow streams for learning and collaboration

The instructional workflow streams we present and analyze in Chapter 4 have great potential for learning and collaboration. For teams in industry and academia that are participating in remote-work, if each worker created these as they worked, they could increase the efficiency of collaboration by affording a type of pseudo pair-programming effect when following along with a coworkers’ workflow to understand or learn from it better.

For example, if Dan the data scientist shares his data analysis with his co-workers, but they don’t understand the context of some of the decisions Dan had made, they might watch an instructional workflow stream of Dan doing his analysis to get this context. For less-experienced team members, the instructional workflow streams generated by more experienced coworkers like Dan can help them gain insights and skills they might otherwise only gain by working and struggling along over long periods of time. Revealing the gory details of complex technical workflows can prepare these novices more on their journey to gain skills to perform them. Yet more work needs to be done to capture these, and to prevent too much ‘noise’ as the length and amount of content might still be a barrier to content that is easy to watch and understand.

Instructional workflow streams can inform designers and researchers who want to create better tools and processes for various technical workflows like data science, programming, digital art, and gaming. These streams show all of the slips and mistakes made by users, and how these tools and processes fail to assist users in avoiding them. If instructional workflow streams were created more often as artifacts produced by completing a workflow, designers and researchers would have a plethora of data to leverage when informing novel solutions for these users.

However, more needs to be done to enable this form of informative streaming. Thinking-aloud might not be the normal process for Dan the data scientist, so a future tool should listen for unique and potentially frustrating occurrences within Dan’s workflow so it can prompt him to give his insights about the event and how he is going to handle it. These events could then be categorized and searched by viewers of Dan’s instructional workflow stream so that they can quickly find where these events they are interested in happen, without having to watch the entirety of Dan’s data analysis. However, these users could still be able to see the context that this event happened for Dan, which he might have cut out in a polished video about his work.

6.2.3 Leveraging design spaces to design spaces

Understanding the design space that currently exists within streaming setups can help potential streamers in their journey to becoming one, but also help designers in discovering gaps that need solutions to enable streamers to acquire and maintain setups. A list of items is not enough, but the space is filled with insider tips and tricks shared by the creators in our videos can help hack your way to a working setup within the space when budget or access prevents you. How do we leverage both this design space and the hidden knowledge within?

So, when Dan the data scientist wants to begin his journey as a streamer so he can show how he navigates his complex technical workflows off to an audience, he is a bit more informed by using the design space we present in Chapter 5. Dan can see the various categories of equipment he needs to invest in. Dan can navigate the design space to make decisions on the type of equipment within the categories to enable him to produce a stream with the amount of fidelity he desires. However, we also presented insights from streamers who detail their own setups and recommendations. This space alone does not yet reflect the complexity of maintaining these systems or the category dependencies on one another. If Dan wants to upgrade his USB microphone to an XLR microphone, he might need an audio interface. Further, specific configurations of the streaming tools he uses and even his environment might be in order due to this upgrade. The work we present discusses these in detail, but there are many advantages for bringing this knowledge into the design space itself as Dan navigates it. For example, if Dan clicks on 'XLR microphone' in some kind of interactive representation of this design space, it might highlight various other parts of the design space he now needs to consider or warn him when his existing setup cannot handle the new equipment. They might also present the configurations that other streamers with setups like his own have, so he might use them as reference for his own setup.

This design space can also inform remote-workers who want to display a professional image to their coworkers during online meetings by using these livestreamers as inspiration. Even though the main goal of these remote-workers might not be to entertain thousands of viewers, improving the quality of their at-home work streaming could increase the fidelity of the connections and interactions at work to similar amounts we once did by being physically in-person within office buildings.

However, there are a few lines of research as future work that I can see extending off of the research in this dissertation to improve Dan's situation even more, which is described in the following section.

6.3 Future work in designing informative systems

As more technical work and training moves online due to pervasive remote work arrangements worldwide, we need tools and experiences that provide transparent workflows to ease tedium found in current workflows and to enable learners to learn from wherever they are, whether it is in person or somewhere across the world. Video production can require a lot of resources and effort, without a team or advanced and costly setups (some content creators have setups that cost over $100,000 USD!) content creators reasonably might feel overwhelmed. We can help content creators by simplifying their workflow through automating a lot of the tedious tasks they usually hire a team for, but allow them to continue to explore and modify the data they are creating to their own preferences.

6.3.1 Intelligent monitoring of tools to produce content for tutorials

First, we can enhance existing workflows for content creators, like programmers who stream by instrumenting their IDEs to prompt the content creator to talk aloud about a problem they are having if they run into an error or start a debugging session to gain important insights for learners who are watching, or even automatically control video production software like OBS to automatically take clips of critical incidents that can later be used to create targeted tutorials using footage they already captured during their normal workflows without having to create new footage to share with others.

For this, I envision a tool called Protege that can monitor the applications being used, like a programmer's IDE or a data scientist's notebook, and begin to build an instructional workflow stream for the user based on interesting and useful events happening within their workflow. If Dan then runs into an error during writing his data analysis code, Protege could take a short video clip of the occurrence. If Dan the data scientist goes to his web browser to confirm the quality of his data, Protege can capture another video clip of this action and Dan's browser, and then organize this clip with similar clips in case Dan would like to produce a targeted tutorial on debugging or web searching strategies for data scientists. Through the detection of these occurrences, Protege might also prompt Dan to think aloud or provide text annotations about what is happening currently in his workflow, so that useful context can be given to any interesting events for Dan's viewers or even himself if he returns to his data analysis and is trying to understand the decisions he made while performing it.

6.3.2 Explorable workflow graphs for "choose your own adventure" tutorials

Once all these video clips from Protege are given annotations and context, we might even be able to build an interactive and explorable graph of Dan's workflow. This might include the main critical path of the workflow but also all of the tangent side paths that happened along the way, since complex workflows tend to have tangents, mistakes, backtracking, and improvised examples. That way, a viewer of Dan's analysis can have a "Choose Your Own Adventure" experience for a tutorial teaching them how to be a data scientist just like Dan. For instance, they can watch the video clips that only comprise the main "critical path" through the graph from start to successful finish of Dan's workflow, or explore clips of tangents represented by side branches if interested or relevant to the viewer. For example, if a viewer comes across an error in their own data analysis, they might check out one of Dan's tangent branches that had him debugging some of his analysis code due to the same error to gain some insights on how to solve the issue themselves. However, if a viewer is well-versed in debugging their code for that specific error, they are free to ignore these tangential branches and focus on whatever content is relevant to them.

6.3.3 Leveraging expert streaming knowledge to guide novices

When these users want to acquire, use, and maintain a streaming setup to livestream their workflows to viewers, we can consider ways to leverage real setups within the design space of streaming setups to inform and assist them in becoming a one-person production crew and have the quality of a livestream that they desire. One way we might do this is through recommendations for users to traverse the design space of livestream setups when acquiring new hardware and provide tips and tricks of alternative solutions based on their need, budget, and goals. So if Dan wants to improve his audio by upgrading his microphone, we can present microphones within his budget that are commonly used by other streamers, provide examples of their use and fidelity, explain the settings and configurations done by these streamers which include the other parts of their setup and how to integrate this upgrade, and then finally inform Dan of other requirements that this upgrade may introduce within the other parts of the design space. Once Dan has a setup he is happy with, we can continue to provide tips, tricks, and other inside knowledge that is being generated by streamers with similar setups as they are used, so that Dan can also leverage this information for his own setup, and perhaps share the same with others.

6.4 Final Thoughts

Adding intelligent functionality can provide amazing benefits for users of the tools that leverage it, such as accomplishing tasks within their workflow quickly and with less required effort. However, we must be careful to not hide the actual work being done since there are many valuable things to be learned from being able to explore the nitty-gritty details of a complex workflow. By monitoring and leveraging existing expert workflows and identifying critical knowledge, even if that knowledge is mostly tacit, we can thoughtfully provide new interactions to assist users effectively but still keep access to elements with potential educational value. We can make it easier for experts to complete, teach, and share complex workflows, and by doing so, we enable novices to learn from them.

Bibliography

[1] Obs: Open broadcaster software. https://obsproject.com/, 2020. Accessed: 2022-02-11.

[2] Abdulaziz Alaboudi and Thomas D. LaToza. An exploratory study of live-streamed programming. In *2019 IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2019, Memphis, Tennessee, USA, October 14-18, 2019*, VL/HCC '19, pages 5–13. IEEE Computer Society, 2019.

[3] Apache. Zeppelin, 2019.

[4] Boone Ashworth. How to host a virtual watch party. WIRED – https://www.wired.com/story/how-to-host-a-virtual-watch-party/, 2020. Accessed: 2021-04-15.

[5] Alan F. Blackwell. Your wish is my command. chapter SWYN: A Visual Representation for Regular Expressions, pages 245–270. 2001.

[6] Johanna Brewer, Morgan Romine, and T. L. Taylor. *Inclusion at Scale: Deploying a Community-Driven Moderation Intervention on Twitch*, page 757–769. Association for Computing Machinery, New York, NY, USA, 2020.

[7] Jon Brodkin. Twitch explains confusing copyright crackdown, urges users to delete videos. https://arstechnica.com/tech-policy/2020/11/twitch-explains-confusing-copyright-crackdown-urges-users-to-delete-videos/, 2020. Accessed: 2021-04-15.

[8] Frederik Brudy, Christian Holz, Roman Rädle, Chi-Jui Wu, Steven Houben, Clemens Nylandsted Klokmose, and Nicolai Marquardt. *Cross-Device Taxonomy: Survey, Opportunities and Challenges of Interactions Spanning Across Multiple Devices*, page 1–28. Association for Computing Machinery, New York, NY, USA, 2019.

[9] Vincent Burel. Voicemeeter potato ultimate mixer. https://vb-audio.com/Voicemeeter/potato.htm, 2022. Accessed: 2022-02-11.

[10] Jie Cai and Donghee Yvette Wohn. After violation but before sanction: Understanding volunteer moderators' profiling processes toward violators in live streaming communities. *Proc. ACM Hum.-Comput. Interact.*, 5(CSCW2), oct 2021.

[11] Carbide. Carbide alpha, 2019.

[12] Sarah E. Chasins, Maria Mueller, and Rastislav Bodik. Rousillon: Scraping distributed hierarchical web data. In *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*, UIST '18, pages 963–975, 2018.

[13] Di (Laura) Chen, Dustin Freeman, and Ravin Balakrishnan. Integrating multimedia tools to enrich interactions in live streaming for language learning. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, page 1–14, New York, NY, USA, 2019. Association for Computing Machinery.

[14] Xinyue Chen, Si Chen, Xu Wang, and Yun Huang. "i was afraid, but now i enjoy being a streamer!": Understanding the challenges and prospects of using live streaming for online education. *Proc. ACM Hum.-Comput. Interact.*, 4(CSCW3), January 2021.

[15] Yan Chen, Walter S. Lasecki, and Tao Dong. Towards supporting programming education at scale via live streaming. *Proc. ACM Hum.-Comput. Interact.*, 4(CSCW3), January 2021.

[16] Zhilong Chen, Hancheng Cao, Yuting Deng, Xuan Gao, Jinghua Piao, Fengli Xu, Yu Zhang, and Yong Li. Learning from home: A mixed-methods analysis of live streaming based remote education experience in chinese colleges during the covid-19 pandemic. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21, New York, NY, USA, 2021. Association for Computing Machinery.

[17] Pei-Yu Chi, Sally Ahn, Amanda Ren, Mira Dontcheva, Wilmot Li, and Björn Hartmann. Mixt: Automatic generation of step-by-step mixed media tutorials. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology*, UIST '12, page 93–102, New York, NY, USA, 2012. Association for Computing Machinery.

[18] Pei-Yu Chi, Joyce Liu, Jason Linder, Mira Dontcheva, Wilmot Li, and Bjoern Hartmann. Democut: Generating concise instructional videos for physical demonstrations. In *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology*, UIST '13, page 141–150, New York, NY, USA, 2013. Association for Computing Machinery.

[19] John Joon Young Chung, Hijung Valentina Shin, Haijun Xia, Li-yi Wei, and Rubaiat Habib Kazi. Beyond show of hands: Engaging viewers via expressive and scalable visual communication in live streaming. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21, New York, NY, USA, 2021. Association for Computing Machinery.

[20] Anna T. Cianciolo and Robert J. Sternberg. *Practical Intelligence and Tacit Knowledge: An Ecological View of Expertise*, page 770–792. Cambridge Handbooks in Psychology. Cambridge University Press, 2 edition, 2018.

[21] Allan Collins, John Seely Brown, and Ann Holum. Cognitive apprenticeship: Making thinking visible. *American Educator*, 15(3):6–11, 1991.

[22] Allan Collins, John Seely Brown, and Susan E. Newman. Cognitive apprenticeship: Teaching the crafts of reading, writing, and mathematics. *Knowing, learning, and instruction: Essays in honor of Robert Glaser*, pages 453–494, 1989.

[23] Juliet M. Corbin and Anselm L. Strauss. *Basics of qualitative research: techniques and procedures for developing grounded theory.* SAGE Publications, Inc., 2008.

[24] Andrew Cross, Mydhili Bayyapunedi, Dilip Ravindran, Edward Cutrell, and William Thies. Vidwiki: Enabling the crowd to improve the legibility of online educational videos. In *Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work & Social Computing*, CSCW '14, page 1167–1175, New York, NY, USA, 2014. Association for Computing Machinery.

[25] Tamraparni Dasu and Theodore Johnson. *Exploratory Data Mining and Data Cleaning*. 1 edition, 2003.

[26] Databricks. databricks, 2019.

[27] Robert DeLine, Danyel Fisher, Badrish Chandramouli, Jonathan Goldstein, Michael Barnett, James F Terwilliger, and John Wernsing. Tempe: Live scripting for live data. In *2015 IEEE Symposium on Visual Languages and Human-Centric Computing*, VL/HCC '15, pages 137–141, 2015.

[28] Ian Drosos and Philip J. Guo. Streamers teaching programming, art, and gaming: Cognitive apprenticeship, serendipitous teachable moments, and tacit expert knowledge. In *2021 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 2021.

[29] LLC Ecamm Network. Meett ecamm live. https://www.ecamm.com/, 2022. Accessed: 2022-02-11.

[30] edX blog. Supporting communities of teaching assistants (tas) and students in moocs. https://blog.edx.org/supporting-communities-teaching, 2013. Accessed: 2021-04-15.

[31] Elgato. Elgato cam link 4k. https://www.elgato.com/en/cam-link-4k, 2022. Accessed: 2022-02-11.

[32] Elgato. Elgato stream deck. https://www.elgato.com/en/stream-deck, 2022. Accessed: 2022-02-11.

[33] Adam Enfroy. 7+ best streaming software (for twitch and youtube) 2022. https://www.adamenfroy.com/best-streaming-software/, 2022. Accessed: 2022-02-11.

[34] Renee Engeln. Zoom fatigue is worse when you don't like your face. https://www.psychologytoday.com/us/blog/beauty-sick/202201/zoom-fatigue-is-worse-when-you-dont-your-face, 2022. Accessed: 2022-02-16.

[35] Jonas Eriksson. Stream with tobii ghost, the first eye tracking twitch extension. https://blog.tobii.com/tobii-ghost-twitch-extension-eye-tracking-overlay, 2018. Accessed: 2022-02-11.

[36] Travis Faas, Lynn Dombrowski, Erin Brady, and Andrew Miller. Looking for group: Live streaming programming for small audiences. In *International Conference on Information*, volume 11420 of *Information in Contemporary Society*, pages 117–123. Springer, Cham, 2019.

[37] Travis Faas, Lynn Dombrowski, Alyson Young, and Andrew D. Miller. Watch me code: Programming mentorship communities on twitch.tv. *Proc. ACM Hum.-Comput. Interact.*, 2(CSCW), November 2018.

[38] Yu Feng, Ruben Martins, Jacob Van Geffen, Isil Dillig, and Swarat Chaudhuri. Component-based synthesis of table consolidation and transformation tasks from examples. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '17, pages 422–436, 2017.

[39] C. Ailie Fraser, Joy O. Kim, Hijung Valentina Shin, Joel Brandt, and Mira Dontcheva. Temporal segmentation of creative live streams. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, CHI '20, page 1–12, New York, NY, USA, 2020. Association for Computing Machinery.

[40] C. Ailie Fraser, Joy O. Kim, Alison Thornsberry, Scott Klemmer, and Mira Dontcheva. Sharing the studio: How creative livestreaming can inspire, educate, and engage. In *Proceedings of the 2019 on Creativity and Cognition*, C&C '19, page 144–155, New York, NY, USA, 2019. Association for Computing Machinery.

[41] C. Ailie Fraser, Tricia J. Ngoon, Mira Dontcheva, and Scott Klemmer. Replay: Contextually presenting learning videos across software applications. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, page 1–13, New York, NY, USA, 2019. Association for Computing Machinery.

[42] Jonas Frich, Lindsay MacDonald Vermeulen, Christian Remy, Michael Mose Biskjaer, and Peter Dalsgaard. *Mapping the Landscape of Creativity Support Tools in HCI*, page 1–18. Association for Computing Machinery, New York, NY, USA, 2019.

[43] GitHub. GitHub Copilot, Your AI pair programmer. https://copilot.github.com/, 2022. Accessed: 2022-04-15.

[44] Google. Openrefine, 2019.

[45] Christopher Grayson. Variety streamers on twitch. https://www.streamscheme.com/variety-streamer-guide/, 2021. Accessed: 2022-02-16.

[46] T. R. G. Green. Cognitive dimensions of notations. In *Proceedings of the Fifth Conference of the British Computer Society, Human-Computer Interaction Specialist Group on People and Computers V*, pages 443–460, 1989.

[47] Tovi Grossman, Justin Matejka, and George Fitzmaurice. Chronicle: Capture, exploration, and playback of document workflow histories. In *Proceedings of the 23nd Annual ACM Symposium on User Interface Software and Technology*, UIST '10, page 143–152, New York, NY, USA, 2010. Association for Computing Machinery.

[48] Sumit Gulwani. Automating string processing in spreadsheets using input-output examples. In *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '11, pages 317–330, 2011.

[49] Sumit Gulwani. Synthesis from examples: Interaction models and algorithms. In *Proceedings of the 2012 14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, SYNASC '12, pages 8–14, 2012.

[50] Sumit Gulwani, William R. Harris, and Rishabh Singh. Spreadsheet data manipulation using examples. *Commun. ACM*, 55(8):97–105, August 2012.

[51] Sumit Gulwani and Mark Marron. Nlyze: Interactive programming by natural language for spreadsheet data analysis and manipulation. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, pages 803–814, 2014.

[52] Sumit Gulwani, Kunal Pathak, Arjun Radhakrishna, Ashish Tiwari, and Abhishek Udupa. Quantitative Programming by Examples. *arXiv e-prints*, page arXiv:1909.05964, Sep. 2019.

[53] Philip Guo. Data science workflow: Overview and challenges. Blog @ Communications of the ACM, Oct 2013.

[54] Philip J. Guo, Sean Kandel, Joseph M. Hellerstein, and Jeffrey Heer. Proactive wrangling: Mixed-initiative end-user programming of data transformation scripts. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, pages 65–74, 2011.

[55] Philip J. Guo, Juho Kim, and Rob Rubin. How video production affects student engagement: An empirical study of mooc videos. In *Proceedings of the First ACM Conference on Learning @ Scale Conference*, L@S '14, pages 41–50, New York, NY, USA, 2014. ACM.

[56] Lassi Haaranen. Programming as a performance: Live-streaming and its implications for computer science education. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '17, page 353–358, New York, NY, USA, 2017. Association for Computing Machinery.

[57] William A. Hamilton, Oliver Garretson, and Andruid Kerne. *Streaming on Twitch: Fostering Participatory Communities of Play within Live Mixed Media*, page 1315–1324. CHI '14. Association for Computing Machinery, New York, NY, USA, 2014.

[58] William R. Harris and Sumit Gulwani. Spreadsheet table transformations from examples. In *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '11, pages 317–328, 2011.

[59] Hideo Hattori. autopep8, 2019.

[60] Yeye He, Kris Ganjam, Kukjin Lee, Yue Wang, Vivek Narasayya, Surajit Chaudhuri, Xu Chu, and Yudian Zheng. Transform-data-by-example (tde): Extensible data transformation in excel. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, pages 1785–1788, 2018.

[61] Zorah Hilvert-Bruce, James T Neill, Max Sjöblom, and Juho Hamari. Social motivations of live-streaming viewer engagement on twitch. *Computers in Human Behavior*, 84:58–67, 2018.

[62] Suz Hinton. Lessons from my first year of live coding on twitch. https://www.freecodecamp.org/news/lessons-from-my-first-year-of-\ live-coding-on-twitch-41a32e2f41c1/, 2017. Accessed: 2022-01-15.

[63] Suz Hinton, Adam Stacoviak, and Jerod Santo. Live coding open source on twitch, March 2018. Accessed: 2022-01-15.

[64] Joseph A. Horvath, Jennifer Hedlund, Scott Snook, George B. Forsythe, and Robert J. Sternberg. *Tacit Knowledge in Military Leadership: Some Research Products and Their Applications to Leadership Development*, volume 1081. US Army Research Institute for the Behavioral and Social Sciences, 1998.

[65] Mu Hu, Mingli Zhang, and Yu Wang. Why do audiences choose to keep watching on live video streaming platforms? an explanation of dual identification framework. *Computers in Human Behavior*, 75(C):594–606, October 2017.

[66] Jetbrains. Datalore, 2019.

[67] Zhongjun Jin, Michael R. Anderson, Michael Cafarella, and H. V. Jagadish. Foofah: Transforming data by example. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, pages 683–698, 2017.

[68] Joonyoung Jun, Woosuk Seo, Jihyeon Park, Subin Park, and Hyunggu Jung. Exploring the experiences of streamers with visual impairments. *Proc. ACM Hum.-Comput. Interact.*, 5(CSCW2), oct 2021.

[69] Jupyter. Jupyter notebook, 2019.

[70] Kaggle. Titanic: Machine learning from disaster. Dataset at https://www.kaggle.com/c/titanic, 2012. Accessed: 2021-04-15.

[71] Niranjan Kamat, Eugene Wu, and Arnab Nandi. Trendquery: A system for interactive exploration of trends. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*, HILDA '16, pages 12:1–12:4, 2016.

[72] Sean Kandel, Jeffrey Heer, Catherine Plaisant, Jessie Kennedy, Frank van Ham, Nathalie Henry Riche, Chris Weaver, Bongshin Lee, Dominique Brodbeck, and Paolo Buono. Research directions in data wrangling: Visualizations and transformations for usable and credible data. *Information Visualization*, 10(4):271–288, October 2011.

[73] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. Wrangler: Interactive visual specification of data transformation scripts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 3363–3372, 2011.

[74] Sean Kandel, Andreas Paepcke, Joseph M. Hellerstein, and Jeffrey Heer. Enterprise data analysis and visualization: An interview study. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2917–2926, December 2012.

[75] Mary Beth Kery, Marissa Radensky, Mahima Arya, Bonnie E. John, and Brad A. Myers. The story in the notebook: Exploratory data science using a literate programming tool. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, pages 174:1–174:11, 2018.

[76] Juho Kim, Philip J. Guo, Carrie J. Cai, Shang-Wen (Daniel) Li, Krzysztof Z. Gajos, and Robert C. Miller. Data-driven interaction techniques for improving navigation of educational videos. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, UIST '14, page 563–572, New York, NY, USA, 2014. Association for Computing Machinery.

[77] Juho Kim, Phu Tran Nguyen, Sarah Weir, Philip J. Guo, Robert C. Miller, and Krzysztof Z. Gajos. Crowdsourcing step-by-step information extraction to enhance existing how-to videos. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, page 4017–4026, New York, NY, USA, 2014. Association for Computing Machinery.

[78] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, Carol Willing, and Jupyter development team. Jupyter notebooks - a publishing format for reproducible computational workflows. In Fernando Loizides and Birgit Scmidt, editors, *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pages 87–90. IOS Press, 2016.

[79] Konstantin Kobs, Albin Zehe, Armin Bernstetter, Julian Chibane, Jan Pfister, Julian Tritscher, and Andreas Hotho. Emote-controlled: Obtaining implicit viewer feedback through emote-based sentiment analysis on comments of popular twitch.tv channels. *Trans. Soc. Comput.*, 3(2), April 2020.

[80] Alice Y. Kolb and David A. Kolb. The learning way: Meta-cognitive aspects of experiential learning. *Simulation & Gaming*, 40(3):297–327, 2009.

[81] Tim Kraska. Northstar: An interactive data science system. *Proceedings of the VLDB Endowment*, 11(12):2150–2164, August 2018.

[82] Sam Lau, Ian Drosos, Julia M. Markel, and Philip J. Guo. The design space of computational notebooks: An analysis of 60 systems in academia and industry. In *2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 2020.

[83] Vu Le and Sumit Gulwani. Flashextract: A framework for data extraction by examples. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '14, pages 542–553, 2014.

[84] Pascal Lessel, Alexander Vielhauer, and Antonio Krüger. Expanding video game livestreams with enhanced communication channels: A case study. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, page 1571–1576, New York, NY, USA, 2017. Association for Computing Machinery.

[85] Jie Li, Xinning Gui, Yubo Kou, and Yukun Li. Live streaming as co-performance: Dynamics between center and periphery in theatrical engagement. *Proc. ACM Hum.-Comput. Interact.*, 3(CSCW), nov 2019.

[86] Lingyuan Li, Jirassaya Uttarapong, Guo Freeman, and Donghee Yvette Wohn. Spontaneous, yet studious: Esports commentators' live performance and self-presentation practices. *Proc. ACM Hum.-Comput. Interact.*, 4(CSCW2), oct 2020.

[87] Gavan Lintern, Brian Moon, Gary Klein, and Robert R. Hoffman. *Eliciting and Representing the Knowledge of Experts*, page 165–191. Cambridge Handbooks in Psychology. Cambridge University Press, 2 edition, 2018.

[88] Ching Liu, Juho Kim, and Hao-Chuan Wang. Conceptscape: Collaborative concept mapping for video learning. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, page 1–12, New York, NY, USA, 2018. Association for Computing Machinery.

[89] Steve Lohr. For big-data scientists, 'janitor work' is key hurdle to insights. Article @ The New York Times, Aug 2014.

[90] StudioCoast Pty Ltd. vmix live production streaming software. https://www.vmix.com/, 2022. Accessed: 2022-02-11.

[91] Zhicong Lu, Michelle Annett, Mingming Fan, and Daniel Wigdor. "i feel it is my responsibility to stream": Streaming and engaging with intangible cultural heritage through livestreaming. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, page 1–14, New York, NY, USA, 2019. Association for Computing Machinery.

[92] Zhicong Lu, Michelle Annett, and Daniel Wigdor. Vicariously experiencing it all without going outside: A study of outdoor livestreaming in china. *Proc. ACM Hum.-Comput. Interact.*, 3(CSCW), nov 2019.

[93] Zhicong Lu, Seongkook Heo, and Daniel J. Wigdor. Streamwiki: Enabling viewers of knowledge sharing live streams to collaboratively generate archival documentation for effective in-stream and post hoc learning. *Proc. ACM Hum.-Comput. Interact.*, 2(CSCW), November 2018.

[94] Zhicong Lu, Rubaiat Habib Kazi, Li-yi Wei, Mira Dontcheva, and Karrie Karahalios. Streamsketch: Exploring multi-modal interactions in creative live streams. *Proc. ACM Hum.-Comput. Interact.*, 5(CSCW1), apr 2021.

[95] Zhicong Lu, Chenxinran Shen, Jiannan Li, Hong Shen, and Daniel Wigdor. More kawaii than a real-person live streamer: Understanding how the otaku community engages with and perceives virtual youtubers. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21, New York, NY, USA, 2021. Association for Computing Machinery.

[96] Zhicong Lu, Haijun Xia, Seongkook Heo, and Daniel Wigdor. You watch, you give, and you engage: A study of live streaming practices in china. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, page 1–13, New York, NY, USA, 2018. Association for Computing Machinery.

[97] Mufan Luo, Tiffany W. Hsu, Joon Sung Park, and Jeffrey T. Hancock. Emotional amplification during live-streaming: Evidence from comments during and after news events. *Proc. ACM Hum.-Comput. Interact.*, 4(CSCW1), may 2020.

[98] Laura MacLeod, Andreas Bergen, and Margaret-Anne Storey. Documenting and sharing software knowledge using screencasts. *Empirical Software Engineering*, 22(3):1478–1507, 2017.

[99] Laura MacLeod, Margaret-Anne Storey, and Andreas Bergen. Code, camera, action: How software developers document and share program knowledge using youtube. In *2015 IEEE 23rd International Conference on Program Comprehension*, pages 104–114. IEEE, 2015.

[100] Jim R. Macnamara. Media content analysis: Its uses, benefits and best practice methodology. *Asia Pacific Public Relations Journal*, 6(1):1, 2005.

[101] Keri Mallari, Spencer Williams, and Gary Hsieh. Understanding analytics needs of video game streamers. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21, New York, NY, USA, 2021. Association for Computing Machinery.

[102] Julia M. Markel and Philip J. Guo. Designing the future of experiential learning environments for a post-covid world: A preliminary case study. In *Symposium on the New Future of Work*, NFW '20, Aug 2020.

[103] Vivien Marx. The big challenges of big data. *Nature*, 498:255–260, Jun 2013.

[104] MasterClass. Serena williams teaches tennis. https://www.masterclass.com/classes/serena-williams-teaches-tennis, 2020. Accessed: 2021-04-15.

[105] Microsoft. Prose sdk, 2019.

[106] Toni-Jan Keith Monserrat, Shengdong Zhao, Kevin Mcgee, and Anshul Vikram Pandey. Notevideo: Facilitating navigation of blackboard-style lecture videos. In *CHI '13 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '13, page 2897–2898, New York, NY, USA, 2013. Association for Computing Machinery.

[107] Mozilla. Iodide, 2019.

[108] Alok Mysore and Philip J. Guo. Torta: Generating mixed-media gui and command-line app tutorials using operating-system-wide activity tracing. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, UIST '17, page 703–714, New York, NY, USA, 2017. Association for Computing Machinery.

[109] Michael Nebeling, Shwetha Rajaram, Liwei Wu, Yifei Cheng, and Jaylin Herskovitz. Xrstudio: A virtual production and live streaming system for immersive instructional experiences. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21, New York, NY, USA, 2021. Association for Computing Machinery.

[110] Observable. Observable, 2019.

[111] obsproject. Websocket api for obs studio. https://github.com/obsproject/obs-websocket, 2022. Accessed: 2022-02-11.

[112] pandas-dev. The pandas project, 2019.

[113] Anthony J. Pellicone and June Ahn. The game of performing play: Understanding streaming as cultural production. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, page 4863–4874, New York, NY, USA, 2017. Association for Computing Machinery.

[114] Kevin P. Pfeil, Neeraj Chatlani, Joseph J. LaViola, and Pamela Wisniewski. Bridging the socio-technical gaps in body-worn interpersonal live-streaming telepresence through a critical review of the literature. *Proc. ACM Hum.-Comput. Interact.*, 5(CSCW1), apr 2021.

[115] Michael Polanyi. *The Tacit Dimension*. University of Chicago Press, 1966.

[116] Suporn Pongnumkul, Mira Dontcheva, Wilmot Li, Jue Wang, Lubomir Bourdev, Shai Avidan, and Michael F. Cohen. Pause-and-play: Automatically linking screencast video tutorials with applications. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, page 135–144, New York, NY, USA, 2011. Association for Computing Machinery.

[117] Quantopian. Qgrid, 2019.

[118] Quora. What is “sameface syndrome” in certain anime artists? https://www.quora.com/What-is-sameface-syndrome-in-certain-anime-artists, 2019. Accessed: 2021-04-15.

[119] Vignesh Ramachandran. Stanford researchers identify four causes for ‘zoom fatigue’ and their simple fixes. https://news.stanford.edu/2021/02/23/four-causes-zoom-fatigue-solutions/, 2021. Accessed: 2022-02-16.

[120] Reddit. YouTube content having better discoverability than just pure Twitch. https://www.reddit.com/r/Twitch/comments/fc3e6u/youtube_content_having_better_discoverability/, 2020. Accessed: 2021-04-15.

[121] David Robinson. Tidy tuesday r screencasts. https://www.youtube.com/playlist?list=PL19ev-r1GBwkuyiwnxoHTRC8TTqP8OEi8, 2020. Accessed: 2021-04-15.

[122] Logitech Services S.A. Streamlabs: All-in-one live streaming software. https://streamlabs.com/, 2022. Accessed: 2022-02-11.

[123] Benjamin Saunders, Julius Sim, Tom Kingstone, Shula Baker, Jackie Waterfield, Bernadette Bartlam, Heather Burroughs, and Clare Jinks. Saturation in qualitative research: exploring its conceptualization and operationalization. *Quality & quantity*, 52(4), 2018.

[124] Donald A. Schon. *The reflective practitioner: How professionals think in action*, volume 5126. Basic books, 1984.

[125] Edward Segel and Jeffrey Heer. Narrative visualization: Telling stories with data. *IEEE transactions on visualization and computer graphics*, 16(6):1139–1148, 2010.

[126] Stephen Tsung-Han Sher and Norman Makoto Su. Speedrunning for charity: How donations gather around a live streamed couch. *Proc. ACM Hum.-Comput. Interact.*, 3(CSCW), November 2019.

[127] Hijung Valentina Shin, Floraine Berthouzoz, Wilmot Li, and Frédo Durand. Visual transcripts: Lecture notes from blackboard-style lecture videos. *ACM Trans. Graph.*, 34(6), October 2015.

[128] Rishabh Singh and Sumit Gulwani. Learning semantic string transformations from examples. *Proceedings of the VLDB Endowment*, 5(8):740–751, April 2012.

[129] Rishabh Singh and Sumit Gulwani. Synthesizing number transformations from input-output examples. In *Computer Aided Verification*, pages 634–651, 2012.

[130] Max Sjöblom and Juho Hamari. Why do people watch others play video games? an empirical study on the motivations of twitch users. *Computers in Human Behavior*, 75(C):985–996, 2017.

[131] Max Sjöblom, Maria Törhönen, Juho Hamari, and Joseph Macey. Content structure is king: An empirical study on gratifications, game genres and content type on twitch. *Computers in Human Behavior*, 73:161–171, 2017.

[132] Sara Stoudt, Valeri N. Vasquez, and Ciera C. Martinez. Principles for data analysis workflows, 2020.

[133] Alina Striner, Andrew M. Webb, Jessica Hammer, and Amy Cook. Mapping design spaces for audience participation in game live streaming. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21, New York, NY, USA, 2021. Association for Computing Machinery.

[134] John C. Tang, Gina Venolia, and Kori M. Inkpen. Meerkat and periscope: I stream, you stream, apps stream for live streams. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16, page 4770–4780, New York, NY, USA, 2016. Association for Computing Machinery.

[135] Lucia Terrenghi, Aaron Quigley, and Alan Dix. A taxonomy for and analysis of multi-person-display ecosystems. *Personal Ubiquitous Comput.*, 13(8):583–598, nov 2009.

[136] Shara Tibken. From ps5 to ford f-150: How a global chip shortage is 'impacting everything'. https://www.cnet.com/tech/mobile/from-ps5-to-ford-f-150-how-a-global-chip-shortage-is-impacting-everything/, 2021. Accessed: 2022-02-11.

[137] Trifacta. Wrangler, 2019.

[138] Twilio. Twilio: Rest apis. https://www.twilio.com/docs/usage/api, 2020. Accessed: 2021-04-15.

[139] Twitch. Extensions: A revolution in live streaming. https://www.twitch.tv/p/en/extensions/, 2021. Accessed: 2022-02-11.

[140] Twitch. Twitch extension dashboard. https://dashboard.twitch.tv/extensions, 2021. Accessed: 2022-02-11.

[141] Dennis Wang, Yi-Chieh Lee, and Wai-Tat Fu. "i love the feeling of being on stage, but i become greedy": Exploring the impact of monetary incentives on live streamers' social interactions and streaming content. *Proc. ACM Hum.-Comput. Interact.*, 3(CSCW), November 2019.

[142] Sarah Weir, Juho Kim, Krzysztof Z. Gajos, and Robert C. Miller. Learnersourcing subgoal labels for how-to videos. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*, CSCW '15, page 405–416, New York, NY, USA, 2015. Association for Computing Machinery.

[143] Hadley Wickham. Tidy data. *Journal of Statistical Software, Articles*, 59(10):1–23, 2014.

[144] Wikipedia. Zoom fatigue. https://en.wikipedia.org/wiki/Zoom_fatigue, 2022. Accessed: 2022-02-16.

[145] Wildbit. Beanstalk: A complete development workflow. https://beanstalkapp.com/, 2020. Accessed: 2021-04-15.

[146] Tom P. Wilson and Cameron Summerson. What is an xlr microphone, and why would i want one? https://www.howtogeek.com/404747/what-is-an-xlr-microphone-and-why-do-i-want-one/, 2019. Accessed: 2022-02-11.

[147] Jacob O. Wobbrock and Julie A. Kientz. Research contributions in human-computer interaction. *Interactions*, 23(3):38–44, apr 2016.

[148] Donghee Yvette Wohn. Volunteer moderators in twitch micro communities: How they get involved, the roles they play, and the emotional labor they experience. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, page 1–13, New York, NY, USA, 2019. Association for Computing Machinery.

[149] Donghee Yvette Wohn, Guo Freeman, and Caitlin McLaughlin. Explaining viewers' emotional, instrumental, and financial support provision for live streamers. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, page 1–13, New York, NY, USA, 2018. Association for Computing Machinery.

[150] Navid Yaghmazadeh, Xinyu Wang, and Isil Dillig. Automated migration of hierarchical data to relational tables using programming-by-example. *Proceedings of the VLDB Endowment*, 11(5):580–593, January 2018.

[151] Saelyne Yang, Changyoon Lee, Hijung Valentina Shin, and Juho Kim. Snapstream: Snapshot-based interaction in live streaming for visual art. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, CHI '20, page 1–12, New York, NY, USA, 2020. Association for Computing Machinery.

[152] Kuat Yessenov, Shubham Tulsiani, Aditya Menon, Robert C. Miller, Sumit Gulwani, Butler Lampson, and Adam Kalai. A colorful approach to text processing by example. In *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology*, UIST '13, pages 495–504, 2013.

[153] Xiong Zhang and Philip J. Guo. Ds.js: Turn any webpage into an example-centric live programming environment for learning data science. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, UIST '17, pages 691–702, 2017.