Software Developers Learning Machine Learning: Motivations, Hurdles, and Desires

Carrie J. Cai
Google Research, Brain Team
Mountain View, CA, USA
cjcai@google.com

Philip J. Guo
UC San Diego
La Jolla, CA, USA
pg@ucsd.edu

***Abstract*—The growing popularity of machine learning (ML) has attracted more software developers to now want to adopt ML into their own practices, through tinkering with and learning from ML framework websites and online code examples. To investigate the motivations, hurdles, and desires of these software developers, we deployed a survey to the website of the TensorFlow.js ML framework. We found via 645 responses that many wanted to learn ML for aspirational reasons rather than for immediate job needs. Critically, developers faced hurdles due to a perceived lack of mathematical and theoretical background. They desired frameworks to provide more basic ML conceptual support, such as a curated corpus of best practices, conceptual tutorials, and a de-mystification of mathematical jargon into practical tips. These findings inform the design of ML frameworks and informal learning resources to broaden the base of people acquiring this increasingly important skill set.**

***Index Terms*—**machine learning, software developers

I. Introduction

Over the past decade, machine learning (ML) has become pervasive across a wide range of domains, from self-driving cars to conversational agents to medicine [1]–[4]. What has fueled this growth is a combination of affordable GPU-powered cloud computing services and modern ML frameworks such as TensorFlow, Keras, Caffe, Theano, and PyTorch [5]. Whereas in the past machine learning was accessible mostly to niche specialists in research labs who wrote their own ad-hoc code, nowadays it is much easier for non ML-specialists to get started by building upon these popular frameworks.

Alongside this growth in ML frameworks has been a parallel growth in interest among software developers who are learning to adopt this new, relatively complex technology. With the increasing availability of open-source ML frameworks and online resources, some developers are learning ML in an *informal self-directed manner*, through directly tinkering with framework code and adapting code examples from official API websites [6]. Many also consult resources such as online programming tutorials [7] and YouTube videos [8].

How do software developers learn and adopt ML so that they can apply it effectively in their own professional practice? In this paper, we sought to understand: a) the motivations of software developers learning to adopt ML, b) the technical and knowledge hurdles they encounter, and c) their specific desires for how modern ML frameworks can better support these self-directed learning needs. We investigated these research questions by deploying a survey to the website of the TensorFlow.js ML framework [9] and analyzing 645 responses. We chose this framework because it was designed to lower barriers to using ML by making it accessible via JavaScript in web browsers [10]; as such, it attracts a broad user base that includes many software developers who are not ML specialists.

Our survey respondents were skilled in software development (web programming in particular) but had little experience with ML. Our analysis found that developers' desires for ML frameworks extended *beyond* simply wanting help with APIs: more fundamentally, they desired guidance on understanding and applying the conceptual underpinnings of ML itself. For example, besides wanting more example code and utilities, many wished for ML frameworks to provide conceptual tutorials, canonical models, and common ML best practices. Along those lines, a substantial portion of respondents felt that their own lack of conceptual and theoretical background was a key hurdle to adopting ML, and desired resources to bridge that gap. Many also indicated that their desire for learning ML was aspirational, rather than a result of immediate programming needs. These motivations and desires suggest that, beyond providing a clear API, ML frameworks may need to fill multiple roles and provide conceptual scaffolding.

Based on these findings, we discuss how ML frameworks could better support software developers' bootstrapping into this new paradigm of computing, such as by de-mystifying esoteric terminology into practical concepts, scaffolding reuse and modification, and providing just-in-time ML best practices within programming workflows. Our findings also raise practical considerations around how ML frameworks can better support learning-by-doing. This work points the way toward the design of integrated learning resources for teaching ML to developers and other non-specialists.

This paper's main contributions are:

- Findings from a survey of 645 respondents, contributing a broad snapshot of software developers in their early stages of learning and applying ML. This paper synthesizes their motivations for learning and adopting ML, the key hurdles they face, and their desires for ML frameworks to better support conceptual understanding.
- Design recommendations for frameworks and learning resources to make ML more accessible to more people.

978-1-7281-0810-0/19/$31.00

II. Related Work

A. ML for Non-Specialists

Although ML is a highly active area of research, there have been relatively few prior studies of how non ML-specialists adopt ML into their practices. Patel et al. interviewed 11 HCI researchers and performed a lab study on 10 CS graduate students who had integrated ML into their prototype systems [11]; they found that although these researchers were experienced programmers, they still had trouble structuring ML workflows, understanding model behavior, and evaluating model performance. Dove et al. surveyed 51 UX designers and found that they perceived ML to be a difficult design material: they had trouble prototyping with ML since it was hard to understand its capabilities and limitations [12]. Similarly, Yang et al. interviewed 13 UX designers who were experienced in incorporating ML into products and found that although they did not have a deep understanding of ML, they created abstractions to help communicate ML-oriented designs with engineers [13]. Our study contributes to this nascent literature via a much larger-scale survey of 645 people, most of whom were software developers rather than UX designers or academic researchers, and on one of the most widely used ML framework platforms.

B. End-User Programming

End-user programming is commonly defined as coding that is done to achieve personal goals rather than to create artifacts for broader public use [14]. Viewed in this light, some of our survey respondents aspire to engage in *end-user ML programming* by incorporating ML into their personal or hobby projects. More generally, our work fits into the longstanding tradition of end-user programming research, which includes both systems [15]–[18] and studies [19]–[22] on making programming more accessible to broader audiences. Our findings point toward the design of lower-barrier ML frameworks that can potentially enable a broader range of programmers to adopt these techniques into their software.

Ko et al. discovered six categories of learning barriers in end-user programming via a study of novice students learning Visual Basic.NET in a university course [22]. Our study findings corroborate some of these barriers (see Key Hurdles section) but makes a novel contribution by uncovering how these barriers manifest within the realm of machine learning.

Relatedly, a study with seven developers found that, in domains that developers are unfamiliar with (e.g., networking protocols), a lack of conceptual knowledge can make it challenging to use API documentation [23]. Those researchers noted that these conceptual aspects of API usability are underresearched, thus deserving of more attention [23]. Our findings build on this early work by uncovering the unique conceptual challenges of programmers adopting ML, and contribute implications for how ML frameworks can better address those knowledge gaps.

C. Informal and Self-Directed Learning

Our respondents were mostly working professionals who chose to learn ML-related programming. In a similar vein, researchers have studied working professionals learning programming in a range of settings, including research scientists [24], web/graphics designers [21], [25], and high school CS teachers [26]. Research has also been conducted on the motivational and practical challenges of informal adult learning, such as a lack of time and confidence [27], [28]. In addition, research on *exploratory programming* identified programmers' needs when learning to do ill-defined exploratory tasks such as data science, revealing the need for fast iteration and backtracking [29]. Among these threads of research, the line of work here that is most relevant to ours studied *conversational programmers* [30]–[32]: non-technical professionals at technology organizations who want to learn some programming to help them communicate better with their engineering colleagues. Like the non-specialist ML learners in our study, conversational programmers have aspirational goals and perceive that knowing a bit about programming may be helpful for their careers, even if they do not need to code in their jobs. Though similar learning challenges may arise, our study specifically examines learners who are already proficient in general programming, but who are new to ML.

III. Methods: Online Survey

To get a broad range of insights from people who are learning or experimenting with ML, we deployed an online survey to users of the TensorFlow.js framework [9]. This is a JavaScript-based ML framework that enables programmers to create neural networks and other models that can run in web browsers. It is an official JavaScript port of the popular TensorFlow system [33] for production-scale ML deployments. We chose this framework in particular because it was explicitly designed to lower the barriers to entry into ML and encourage a broader developer population to try ML [10].

To reach a wide audience, we posted the survey as a link on the home page of the framework's website and also publicized it through its user mailing list and Twitter account. Participation was voluntary; we did not pay survey respondents. We ran this survey for two months during summer 2018. Participants were required to be at least 18 years old.

A. The Survey Instrument

Our survey starts with questions about background and self-reported technical proficiency:

- Which best describes your job role: e.g., web developer, mobile app developer, data scientist, ML engineer, educator, technology leader (etc.), other (open-ended)
- Rate your level of proficiency in: 1) programming, 2) web programming, 3) using existing neural networks[1], 4) developing new neural networks, and 5) using/developing other ML algorithms that are not neural nets. Each

[1]Modern ML frameworks are often used to make neural networks, so we split ML proficiency by neural nets vs. other algorithms.

question presented a 5-point Likert scale from beginner to expert, with an additional choice for "no experience."
- In what ways have you previously tried to learn ML? e.g., online tutorials, books, courses, other (open-ended)

The survey also asked free-response questions, which were divided into two sections. One section captured general learner motivations and hurdles: Why are you interested in learning ML? What hurdles have you faced in trying to learn ML? Another section addressed framework-relevant motivations and desires: What motivated you to try TensorFlow.js? What new features would you like in TensorFlow.js?

B. Data Overview and Analysis

We received 645 total survey responses which contained at least one non-blank open-ended response. To classify the contents of open-ended responses, the research team (2 researchers) first read over all responses. We used an inductive analysis approach [34] to determine a classification scheme for each question as a team. Then each researcher independently coded the responses according to those categories. To assess reliability and measure agreement between coders, we computed the Cohen's Kappa for all four questions. We found substantial agreement between coders, with kappa scores of $\kappa = 0.68, 0.73, 0.73, 0.68$ respectively. Finally, the research team made another full pass through all responses together to resolve any conflicts in labels.

C. Study Design Limitations

Our study was conducted through a single, widely used platform, rather than multiple ML platforms or frameworks, so our respondents may not be representative of all programmers. That said, our approach is consistent with prior papers on learning programming which also surveyed participants through a single widely-used platform to capture overarching trends [35], [36]. More importantly, our focus on the JavaScript community has the advantage of providing a snapshot of a community in its early days of ML, in terms of both libraries and ML expertise (rather than a more mature ML community with experienced ML practitioners in, say, Python or C++). Although it is possible that some respondents are already proficient in ML, our data shows that the vast majority do *not* work in ML-related jobs, and that most self-reported limited prior experience with ML. Thus, the choice of platform supports and is consistent with the overall aims of this research.

We deployed this survey to the website of an ML framework, not to an explicit learning resource such as an online course. We deliberately chose this setting in light of the recent growth of ML frameworks being developed to democratize ML to a broader pool of developers, and informed by prior observations that developers often learn opportunistically from example code and documentation [6]. While we focus our recommendations on how ML frameworks specifically can better support non-specialists, our survey may also reflect online learning experiences more broadly, given that some respondents had also attempted to learn from tutorials, help forums, etc. We used a survey to capture a broad snapshot of experiences, but interviews with selected users could also help us gain deeper insights to supplement the breadth achieved by our survey. Finally, although the study achieved a substantial sample size, the size and characteristics of the population being sampled from are unknown, so we cannot rule out the possibility of sampling bias.

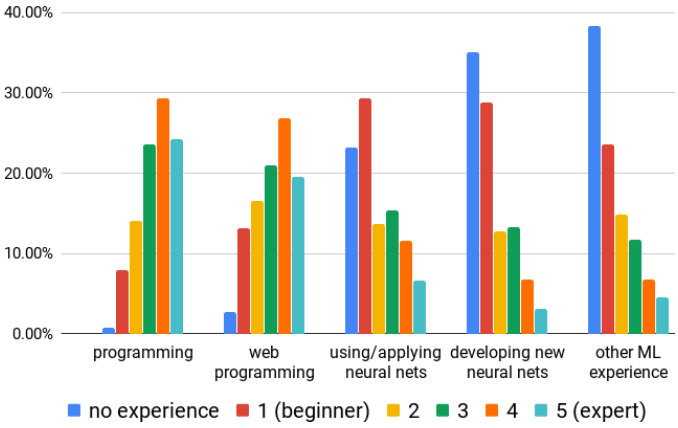

Fig. 1. Levels of programming and ML-related experience (N=644).

IV. Job Roles and Prior Experience

To understand the context of our respondents' perspectives, we first describe their job roles and self-reported skill levels. While some were ML engineers (12%) or ML researchers (5%), the vast majority (83%) of respondents were non ML-specialists. Among these non ML-specialists, most were developers (82%), including web developers, software engineers, mobile app developers, developers in training, and data scientists, with the most common category being web developers. The remaining minority of job roles spanned a variety of fields, such as entrepreneur, product manager, marketer, doctor, maker, artist.

As Figure 1 shows, most respondents were skilled in software development and web programming but inexperienced in machine learning. Specifically, 66% reported limited experience (i.e., skill rating = No Experience, 1, or 2) in using neural networks, 77% had limited experience in developing new neural networks (NN), and 77% in using other (non-NN) machine learning algorithms. In contrast, only a minority reported limited experience in programming (23%) and web programming (33%).

Respondents reported myriad ways in which they tried learning ML. Among those who reported using specific learning resources, a vast majority turned to online tutorials (89%), some took online courses (54%) or accessed online help (52%), and fewer used print documents such as books and magazines (34%). Only a minority had enrolled in formal courses (20%). Some mentioned subscribing to specific educational YouTube channels such as The Coding Train [37]. This prevalence and diversity of online resources indicates that our respondents often participated in informal learning contexts rather than formal classroom settings.

Intellectual Fascination	**26%**
cool and interesting	18%
superhuman intelligence	6%
better than normal programming	2%
ML is "The Future"	**17%**
perception that ML is the future of tech	10%
ML is rapidly rising in prominence	7%
ML Can Improve Human Lives	**14%**
solve critical problems	5%
make the world better	4%
automate mundane tasks	4%
create better user experiences	1%
Develop Career-Enhancing Skills	**14%**
expand technical skillset	7%
generally improve career potential	7%
Specific Application Domains (e.g., NLP)	**14%**
Immediate Need for Job or Hobby	**11%**
Other	**12%**

TABLE I
SUMMARY OF 286 RESPONSES TO "WHY ARE YOU INTERESTED IN LEARNING MACHINE LEARNING?" TOTALS ADD UP TO MORE THAN 100% SINCE WE PLACED SOME RESPONSES INTO MORE THAN ONE CATEGORY.

V. MOTIVATIONS FOR LEARNING ML

Respondents had diverse motivations for learning ML. Table I shows that the most common was that they found the *idea* of ML intellectually fascinating (26%). Other reasons included feeling that ML is "the future" (17%), desire to improve human lives (14%), desire to develop one's career and skill sets (14%), and interest in a specific application area (14%). Only a small portion reported needing ML right now for a specific job task or other use case (11%). Here are the most salient responses:

A. Intellectual Fascination

The most prominently reported interest in ML stemmed from fascination by the perceived superhuman intelligence of ML systems (26%). Some specifically alluded to its capacity to accomplish feats such as the AlphaGo-Zero AI playing Go better than the top humans [38]: *"When I see "Alpha-Zero"...all you need to do is, input rule then it builds up super-human player. Literally, it makes me thrilled then I started learn ML."* Some also contrasted ML to traditional programming, describing how ML transcends the limitations of existing programming languages:*"The core idea of just training a machine and not traditional coding is fascinating."*

B. ML is "The Future"

Respondents also attributed their interest in ML to the growing popularity and perceived future importance of ML (17%). In particular, many described ML as inevitably being the future. For instance, one person stated that *"ML is the star of the coming decade in IT industry,"* and another declared it is the *"strongest thread in the future, beyond nano technology."*

While some were purely future-oriented in their responses, others described ML as being rapidly rising in prominence. For instance, people made remarks such as *"ML seems to be catching everyone's attention"* or *"It's the hot new stuff."* Others pointed to the urgency of its unparalleled growth, stressing that *"it is a huge field which is emerging really fast [...] going to change the computer industry really quickly."*

C. ML Can Improve Human Lives

14% of respondents attributed their interest in ML to its potential for improving human lives. For example, many looked to ML for *"making the world a better place,"* while others wanted *"to develop something which can help our coming generation."* Responses described many humanitarian goals, ranging from the desire to *"get rid of cancer using machine learning,"* to helping *"students interact with their environment through technology."* One pointed to these more worldly goals as a reason for growing beyond their current job: *"I'd like to be involved in some of the more important problem solving in the world. At the moment I'm a frontend web developer and I'm looking for greater challenges."*

Notably, one person desired to learn ML specifically because they felt that ML is not inclusive enough of end users and non-ML domain experts: *"I believe that doctors...need to be more involved in developing AI/ML apps. The current trend seems to be 'AI/ML: better than doctor A-Z' which I believe is not a proper metric. A better more useful metric would be 'AI/ML: actually helps doctors [...] increase accuracy/efficiency to save more lives.'"*

These results suggest that, for many seeking to learn ML, the reasons extend beyond personal or strategic gains to greater social causes. While many described in the prior section were fascinated by the capacity of ML to mimic or even replace human intelligence, the perspectives here reinforce a more human-centric mindset: that automation ultimately serves to better the lives of human beings.

D. Career-Related Motivations

14% of respondents alluded to future career motivations or a desire to expand their skill set. These tended not to be imminent needs in their current job, but rather aspirational desires to gain new skills given a changing job landscape. For example, some expressed that *"as a dev I need to stay relevant,"* whereas others more explicitly cited "*fear of future unemployment*" and "*good salary*" as practical motivations.

Surprisingly, fewer people described needing to use ML in their current jobs, despite many feeling that ML is becoming ubiquitous. Only 11% indicated that they needed ML for a current project. One explanation could be that even production teams are only now starting to experiment with ML. Some described being *"asked to look into it for a summer job,"* or *"we are beginning to explore it at work."* Others felt usage of ML required convincing team-mates to change their mindset: *"always hard to tell the PM [product manager] we can do something with math, not just a if else."* A high barrier to entry may mean that entire teams – not just individuals – could lack the expertise to get started with ML.

Here the desire to learn ML more frequently stems from aspirational goals and perceived future needs than from concrete, near-term challenges. Also, the recent hype and perceived growth of ML may have led many to consider it imminent and relevant, even in the absence of specific use cases.

Lack of Conceptual & Math Understanding	**42%**
don't know math/stats/calculus/linear algebra	25%
don't understand theory / underlying concepts	12%
don't understand ML algorithms	5%
Lack of Resources Bridging Theory & Practice	**21%**
lack of documentation or tutorials	9%
tutorials are too complex	6%
lack of code-based examples	6%
Implementation Challenges	**21%**
architecturing structure of ML model	6%
API/syntax/language issues	6%
latency and performance issues	5%
training and testing are difficult	4%
Challenges in Getting Started	**20%**
don't know where to start / no concrete use case	4%
don't have a dataset	4%
too many choices in tools	4%
no time	4%
installation hurdles	4%
Other	**5%**

TABLE II
SUMMARY OF 232 RESPONSES TO "WHAT HURDLES HAVE YOU FACED IN TRYING TO LEARN ML?" TOTALS ADD UP TO MORE THAN 100% SINCE WE PLACED SOME RESPONSES INTO MORE THAN ONE CATEGORY.

E. Motivations when Selecting a Framework

Of the 421 responses to "What motivated you to try TensorFlow.js?" the three most common categories were: 1) it lowers barriers to getting started (35%), 2) it leverages the browser's run-time environment (28%), and 3) ease of deployment and maintenance (19%). Developers were primarily drawn to the easy-to-install nature of the framework (*"No installations. No version clashes. No virtual envs."*), and to the perception that they could get started with ML with less cognitive overhead (*"I am already familiar with javascript as a web dev [...] the only concepts I will need to grasp are ML concepts."*). The speed with which they could get started may be key, given that learners may desire the freedom to casually try out ML without needing to first invest a large amount of upfront effort: *"If I enjoy it I will take a deeper dive."* Consistent with expectancy-value theory [39], given limited time, casual learners may be more inclined to try something new if the perceived overhead is low. Finally, developers appreciated the possibility of more easily deploying their ML apps to end-users, with few software dependencies: *"Its ability to be used in a web browser, using any device without any environment setup."* In sum, developers may be drawn to frameworks where they can quickly try out ML in a lightweight way and easily deploy or share their prototypes with others.

VI. KEY HURDLES TO LEARNING ML

Table II shows four main categories of hurdles to learning ML; we now describe representative examples from each one.

A. Lack of Conceptual and Mathematical Understanding

By far the most frequently mentioned hurdle to learning ML was a lack of conceptual understanding (42%). Though generally skilled in programming, respondents often encountered a *"steep learning curve of learning the foundation of ML concepts,"* and felt they *"never get [a] full understanding of the algorithm[s]."* Notably, a substantial portion of respondents attributed hurdles to their own lack of math background: many felt they lacked *"linear algebra knowledge," "good calculus background,"* or *"statistical experience."* This perspective may in part be influenced by the prevalence of mathematical terminology in online resources and documentation. For example, respondents described being *"too often faced with mathematical equations (e.g. backward propagation), that are very hard to implement correctly."*

These respondents felt that conceptual hurdles made it hard to progress beyond the initial installation or basic tutorial examples: *"Beginning with the hello world of machine learning to a few tutorials later is hard because [of] the lack of statistical experience. Math behind it."* Thus, even in cases where logistical barriers are reduced (e.g., easier setup in web-based environments), some still perceived a steep learning curve due to math-related concepts.

B. Lack of Resources Bridging Mathematical Theory and Real-World Practice

21% were hindered by a lack of resources bridging theory and code. While some felt that current resources do not provide enough conceptual intuition or theory underlying the code, others desired less theoretical and more code-based examples. Those who desired more theory wanted to *"dive deeper into the background details of algorithm,"* and lamented that *"most tutorials online focus more on [...] just running the code while not highlighting much on the mathematical part."* Others desired less theory: *"I am more of a web dev [...] so I have to be productive and cannot be into all theory around ML,"* and wanted tutorials with *"no crazy calculus as I haven't taken it yet, and ones that actually show code."*

These viewpoints might at first appear contradictory: while some desired more explanations of mathematical theory, others wished for more code and less math. However, both sides reflect a common desire to translate esoteric math jargon into practical concepts that are easier for non-ML-specialists to digest. Interestingly, despite efforts to democratize ML, the math-heavy terminology prevalent in online resources may have inadvertently contributed to imposter syndrome: many felt that current resources are targeted to those who already have an ML background, commenting that they *"cater to the intermediate researcher,"* that there are *"few resources for product management people,"* or that there are *"assumptions made by tutorials of expected knowledge from the reader."* Some had trouble finding tutorials to begin with, and disliked *"having to learn solely through documentation."*

These sentiments may reflect mismatched expectations: by reducing installation and programming overhead, some frameworks have broadened the range of users who can quickly get started with basic examples. Yet, most still lack the conceptual scaffolding necessary to take users beyond those initial steps, thus leading to potential disappointment. This lack of scaffolding is well depicted by one respondent as *"abrupt changes from high-level overviews to low-level code without necessary transitions,"* and by another as *"the gap between math theory and practice."*

C. Implementation Challenges

21% mentioned implementation challenges, the most common being related to structuring the components of an ML model, converting raw data into algorithmic inputs and outputs, and training and testing models.

First, respondents were uncertain how to structure the basic architecture of a model such as a neural net: *"It is just too difficult to implement even a CNN if the user doesn't know its basic structure."* As tutorials tend to offer a limited set of simple examples, people struggled to identify the correct architecture to use for their own specific scenario. As a result, they encountered a range of implementation challenges, such as not knowing *"how many units I have to put in [while] adding layers to the model," "how to apply the suitable activations in each layer,"* and *"deciding what optimizers, loss function etc. to use."* These resemble *coordination* and *use barriers* from Ko et al.'s list [22]. Without a conceptual understanding of which architectures to use in which scenarios, users' level of success tended to depend on whether the code examples mapped well to their desired use case. While many struggled to find relevant examples (*"all demos are using sequential models, [but] I would like to create a model with different input sets."*), those who found relevant ones had a smoother experience (*"The getting started tutorial was similar to my problem so it was a good starting point."*).

Beyond structuring the model itself, programmers found it difficult to wrangle raw data into input/output formats to feed into ML models. One reason for this is that data structures need to first be converted into data objects used in neural nets: people spent time *"going back and forth between user provided values and tensors"* and *"converting plain arrays into tensors."* However, these symptoms may also be indicative of a deeper challenge: developers may not know how to transform a real-world dataset and problem statement into a concrete ML formulation, an instance of a *design barrier* [22].

Aside from structuring the model and transforming data into inputs, people also encountered hurdles training and testing their models, such as not knowing what to do if the model is not performing well (*"How do I debug a program that isn't working?"*) and not understanding what is happening behind the scenes (*"A lot of things are still black boxes, and part of results are indistinguishable from magic."*). These resemble *understanding* and *information barriers* from Ko et al. [22]. Some also found the process of tuning parameters burdensome (e.g., *"Seems like most of the work goes into tweaking parameters"*, *"Too many parameters"*).

D. Challenges in Getting Started

A commonly reported set of hurdles included challenges in even getting to the point where they could start productively writing ML code (20%), due to not knowing what problems to address using ML in the first place. For instance, respondents had trouble *"deciding what aspects and applications of ML to focus upon"*, *"not knowing exactly what kind of projects to work on"*, and more generally, *"what to apply ML to, where ML succeeds, where it sucks."* These reports are consistent with our prior finding that many people desire to learn ML aspirationally and thus may not have a specific use case in mind or even a strategy to determine which problems to tackle first. The decision overhead in getting started can be substantial, given that casual learners may have limited time. One respondent reported: *"Due to the rest of life, I have to fit learning into small 5-15 minute blocks."*

Relatedly, respondents also had trouble finding appropriate data sets. Some simply said they lacked *"interesting data sets to work with."* Others found it hard to *"gather data for [my] own tests"* or felt they *"don't know how to make or use them. Maybe I need to learn Data Science."* Because ML relies on training data at its core, finding and cleaning interesting yet not-too-complex data sets could be challenging, particularly for programmers and web developers who may not be used to wrangling data [16], [40] on a regular basis.

This lack of a concrete starting point is further compounded by an abundance of choices in frameworks and applications. People enumerated hurdles such as *"picking the right ML framework,"* uncertainty over *"what systems to use,"* or generally *"too many possibilities."* These fall under *selection barriers* in Ko et al.'s list [22]. Faced with a large space of options, developers may simply turn to the most popular or convenient frameworks with the lowest perceived overhead. In light of these hurdles, web-based frameworks may have been attractive to JavaScript developers because they reduced some challenges to getting started, such as installation and programming language barriers.

Pre-made Models Demonstrating Best Practices	**22%**
Include Conceptual Tutorials and Examples	**15%**
Visualizations and Visual Programming	**12%**
Cross-framework Compatibility	**12%**
Data Wrangling Utilities	**9%**
Faster Computation	**9%**
Missing specific API function calls	**7%**
Other	**19%**

TABLE III
SUMMARY OF ML FRAMEWORK DESIRES (201 RESPONSES). TOTALS ADD UP TO $>100\%$ SINCE SOME RESPONSES FIT MULTIPLE CATEGORIES.

VII. DESIRES FOR ML FRAMEWORKS

Table III summarizes responses about what new features respondents would like to see in the ML framework. Responses revealed an underlying desire for conceptual scaffolding, *beyond* basic support for using the API: users desired to be taught the implicit ML best practices and concepts that would enable them to effectively apply the framework to their particular problems. Note that some also had desires unrelated to conceptual scaffolding (e.g. faster computation), but those tended to be the minority of respondents who had more ML experience. We now elaborate on the three top categories:

A. Pre-made Models Demonstrating Best Practices

The most frequently reported desire was to provide pre-made ML model architectures that illustrate *best practices* in the field (22%). For instance, some saw using pre-made models as a means for observing well-vetted canonical examples: *"I'd like to be able to use predefined neural net*

architectures that are 'standard' or proven to work for a particular application." Others asked for general ML idioms (*"Show some recommended mythology"*) or industry-standard procedures (*"Are there recommended approaches to follow when productionizing a model?"*). Some simply desired models to work out-of-the-box so that they could easily incorporate them into their application code. While some frameworks do offer pre-made machine learning modules, these resources could be more useful to non-specialists if they came equipped with best practices for *how* to adapt them to example problems, so that users know when and where they can be applied. This could help build conceptual understanding, the lack of which many cited as a key learning hurdle.

B. Include Conceptual Tutorials & Examples in Frameworks

Another common desire was for more conceptual tutorials and examples (15%) to be included into frameworks, beyond API documentation: *"I prefer learning by doing, so I would like to see more tutorials, examples and books..."* Some described how current ML frameworks tend to be disjoint from online learning resources: *"I...start on other ML learning resources (e.g. google results, Andrew Ng coursera) and once I am finished will try to implement ideas in Tensorflow."* To this point, some appreciated when conceptual exercises were embedded directly within the programming workflow: *"It was nice [...] to have the small code demos that you can edit and run right there. Really helps basic understanding."* For theories that are complex and esoteric, developers desired that they be paired with concrete examples to aid understanding: *"More examples in the documentation. I'm an engineer and it's easier to understand examples than math theories."* Overall, these comments reflect a desire to bridge the conceptual gulf between mathematical jargon and code, a key challenge for many non-specialists. In sum, users desired conceptual support to be more tightly interwoven into frameworks.

C. Visualizations and Visual Programming Interfaces

Many respondents also desired run-time visualizations of ML model behavior and, to a lesser extent, visual programming interfaces (12%). Visualizations could help learners build conceptual understanding of a neural net's mechanics, as well as diagnose otherwise opaque problems: For instance, some wanted to see *"a clear visual representation of what the network does"*; others wanted to easily access internal states (*"output and weights of each hidden layer"*) and track progress as the network is being trained. Graphing data such as loss and accuracy could help *"identify if [the] network is too small (not catching up) or too big (overfitting)"*. Although more frameworks are providing visualizations (e.g. Gestalt [41], TensorFlow Graph Visualizer [42]), actively leveraging those visualizations to support conceptual understanding for novices remains an open research problem.

VIII. Discussion and Design Implications

Although frameworks have broadened the range of people who can quickly get started with ML, developers still face many challenges learning and adopting ML into their own practices. Evidently, providing a framework API intended to ease software development is not enough. In light of our findings, we discuss how to redesign ML programming resources to better serve the growing user base of non-specialists.

A. De-mystify Mathematical and Algorithmic Concepts

A substantial portion (42%) of respondents attributed key learning hurdles to their own lack of mathematical and algorithmic background (Table II). Despite ongoing efforts to lower the barriers to ML, and despite a preponderance of ML-newcomers aspiring to learn, surprisingly many of these developers perceived current resources to be intended for more advanced audiences. This impression is in part fueled by the prevalence of esoteric, math-heavy terminology (e.g. ReLU) in code and documentation. To reduce imposter syndrome, future resources could create an intermediate layer of scaffolding that synthesizes theory into digestible, practical concepts. For example, the mathematical details of gradient descent could be abstracted into practical concepts used for tuning a model's learning rate. While some online tutorials do offer such conceptual scaffolding (e.g. The Coding Train, Fast.ai), many framework users were unaware of them. Integrating those resources directly into ML frameworks can help provide conceptual scaffolding during active experimentation.

B. Support Learning-by-Doing

Our survey also revealed that developers desired ML frameworks to teach them not only how to use the API, but also the implicit best practices and concepts that would enable them to effectively apply the framework to their particular problems. Though ML frameworks are not traditional learning platforms, developers are indeed using them as lightweight vehicles for learning and tinkering.

These desires raise important questions: How can ML frameworks better provide informal hands-on learning opportunities? Should ML frameworks coordinate with existing online learning resources, or build their own custom experiences? For example, to support those who would like to learn-by-doing ML without taking an online course, ML frameworks could provide key points in the API where curriculum developers could embed dynamically runnable pedagogical code snippets. One practical challenge is that frameworks may need to satisfy the needs of both advanced ML researchers and newcomers. Thus, for ML education to scale, ML frameworks may need to more clearly distinguish between a spectrum of resources aimed at different levels of expertise, and make explicit the target audience for each. To set appropriate user expectations, for example, frameworks could differentiate between API-specific onboarding and ML onboarding, or explicitly unify the two.

C. Support Re-use and Modification of Pre-made ML Models

A common desire was to have access to libraries of canonical ML models that demonstrate best practices across a diversity of use cases (Table III). Canonical models could enable

users to modify an existing template rather than creating new ones from scratch. Although pre-made models are already used by experts for re-purposing existing models to new domains [43], or for taking output from one neural net as input to a different neural net [44], [45], in their current form, these models are meant for expert use and do not provide support for novice consumption.

In our survey, developers reported substantial hurdles applying and modifying existing examples to their own use cases. Thus, the provision of pre-made ML models should also be coupled with explicit support for *modification*. One way is to provide repositories of model modifications and "remixes" as first-class objects. For example, authors could upload modifications to a centralized gallery (similar to Scratch's Remix [46]). Or, GitHub pull requests for uploading new pre-made models could be standardized to include modification examples. Overall, we view supporting model modification as important future work.

D. Synthesize ML Best Practices into Just-in-time Hints

Interestingly, many novices were drawn to ML with hopes that it could better the world and push the frontiers of intelligence (Table I); such high-level worldly goals tended to be less prominent in people learning regular programming, who instead focused more on task- or career-oriented goals [47]. This observation may be rooted in the paradoxical appeal of ML and recent trends in deep learning [48]: on the surface, they appear to drastically reduce human effort spent hand-engineering features, by discovering program parameters automatically from data. In practice, however, authoring ML requires substantial human intelligence and expertise, involving a myriad of decisions beyond that of regular programming: e.g., which model architectures to use, pre-processing steps to apply to raw data, and hyperparameters to tune. Relative to older ML techniques (e.g., SVMs) where feature selection is vital, in more recent techniques (e.g., deep neural networks [48]) where model tuning is a key challenge, it may be even more bewildering how specific sets of parameters affect model performance. While experts may acquire this folk wisdom through years of dedicated trial and error, the decision overhead can be overwhelming or even demoralizing for casual learners. Furthermore, the hype surrounding ML means that developers may not have a clear sense of which problems are practically feasible for ML to begin with.

To help narrow this broad space of decision possibilities, ML frameworks and resources should embed tips on best practices into the programming workflow, ranging from high-level tips (e.g., which classes of models are suitable for which problems), to low-level strategies (e.g., common diagnostic checks for debugging a model). For example, to give developers a starting point, integrated tools could automatically analyze a user's dataset (e.g., determine its domain) and suggest plausible models. Likewise, ML frameworks could run automatic diagnostic checks on model performance, and surface practical tips (e.g., "decrease learning rate" if the model is not converging), along with an explanation for the underlying concepts behind those tips. These tips could help developers casually become aware of ML idiosyncrasies from within their existing programming workflow. As the frontiers of ML are still evolving, this wisdom can also help both experts as well as novices in creating a shared reservoir of community-curated best practices. As such, it serves as a form of universal design [49], where focusing on a particular user population can result in technologies that benefit everyone.

E. Emphasize and Support the Experimental Nature of ML

Although a stronger conceptual foundation could be helpful, in practice much of modern ML practice also involves dedicated trial and error. Despite the heavy math involved, even ML experts build intuition over time through experimentation, relying on rules of thumb as opposed to pure applications of mathematical theory. In our study, a surprisingly large portion of respondents felt that theoretical understanding was their main limitation. These findings, coupled with existing evidence that novices may expect ML to "just work" as a black box [11], suggest that ML resources ought to emphasize experimental trial and error as a core part of the ML process. Such an emphasis could also help combat imposter syndrome, in cases where a developer may otherwise be discouraged if a model fails to perform as intended.

Likewise, while pre-made models and data pre-processing utilities could initially help programmers easily integrate ML into personal projects, their convenience may also hinder learning, if users never learn how to systematically use trial and error as a way to overcome unexpected model behavior. To scaffold and encourage experimentation, ML resources could offer pre-made models coupled with exercises specifically dedicated to debugging, modifying, and transferring those models to new scenarios. Alternatively, the scaffolding of pre-made models could be gradually removed, so that developers can gradually advance from using pre-made models to modifying or developing new ones.

IX. Conclusion

Software developers are now treating modern ML frameworks as lightweight vehicles for learning and tinkering. Our work provides evidence that, even with the existence of such APIs, developers still face substantial hurdles due to a perceived lack of conceptual and mathematical understanding. In the future, ML frameworks could help by de-mystifying theoretical concepts and synthesizing ML best practices into just-in-time, practical tips. Taken together, this work provides timely implications for how to assist developers in adopting this powerful form of computing into their own practices.

Acknowledgments

We would like to thank Yannick Assogba, Michael Terry, Lauren Hannah-Murphy, and Sandeep Gupta for their thoughtful guidance and feedback on this research. We also thank Nikhil Thorat, Daniel Smilkov, Ann Yuan, Martin Wattenberg, Fernanda Viegas, and the TensorFlow.js team for their support.

REFERENCES

[1] C. J. Cai, E. Reif, N. Hegde, J. Hipp, B. Kim, D. Smilkov, M. Wattenberg, F. Viegas, G. S. Corrado, M. C. Stumpe, and M. Terry, "Human-centered tools for coping with imperfect algorithms during medical decision-making," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. ACM, 2019.

[2] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun, "Dermatologist-level classification of skin cancer with deep neural networks," *Nature*, vol. 542, no. 7639, p. 115, 2017.

[3] A. S. Miner, A. Milstein, S. Schueller, R. Hegde, C. Mangurian, and E. Linos, "Smartphone-based conversational agents and responses to questions about mental health, interpersonal violence, and physical health," *JAMA internal medicine*, vol. 176, no. 5, pp. 619–625, 2016.

[4] C. Urmson *et al.*, "Self-driving cars and the urban challenge," *IEEE Intelligent Systems*, vol. 23, no. 2, pp. 66–68, 2008.

[5] P. Guo, "How did people write machine learning code in the past?" Comm. of the ACM blog, 2018, accessed: 2019-04-01.

[6] J. Brandt, P. J. Guo, J. Lewenstein, M. Dontcheva, and S. R. Klemmer, "Two studies of opportunistic programming: Interleaving web foraging, learning, and writing code," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '09. New York, NY, USA: ACM, 2009, pp. 1589–1598.

[7] A. Mysore and P. J. Guo, "Porta: Profiling software tutorials using operating-system-wide activity tracing," in *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*, ser. UIST '18. New York, NY, USA: ACM, 2018, pp. 201–212. [Online]. Available: http://doi.acm.org/10.1145/3242587.3242633

[8] K. Khandwala and P. J. Guo, "Codemotion: Expanding the design space of learner interactions with computer programming tutorial videos," in *Proceedings of the Fifth Annual ACM Conference on Learning at Scale*, ser. L@S '18. New York, NY, USA: ACM, 2018, pp. 57:1–57:10. [Online]. Available: http://doi.acm.org/10.1145/3231644.3231652

[9] "TensorFlow.js: A JavaScript library for training and deploying ML models in the browser and on Node.js," https://js.tensorflow.org/, 2018, accessed: 2019-04-01.

[10] D. Smilkov, N. Thorat, Y. Assogba, A. Yuan, N. Kreeger, P. Yu, K. Zhang, S. Cai, E. Nielsen, D. Soergel *et al.*, "Tensorflow. js: Machine learning for the web and beyond," *SysML*, 2019.

[11] K. Patel, J. Fogarty, J. A. Landay, and B. Harrison, "Investigating statistical machine learning as a tool for software development," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '08. New York, NY, USA: ACM, 2008, pp. 667–676.

[12] G. Dove, K. Halskov, J. Forlizzi, and J. Zimmerman, "Ux design innovation: Challenges for working with machine learning as a design material," in *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, ser. CHI '17. New York, NY, USA: ACM, 2017, pp. 278–288.

[13] Q. Yang, A. Scuito, J. Zimmerman, J. Forlizzi, and A. Steinfeld, "Investigating how experienced ux designers effectively work with machine learning," in *Proceedings of the 2018 Designing Interactive Systems Conference*, ser. DIS '18. New York, NY, USA: ACM, 2018, pp. 585–596.

[14] A. J. Ko, R. Abraham, L. Beckwith, A. Blackwell, M. Burnett, M. Erwig, C. Scaffidi, J. Lawrance, H. Lieberman, B. Myers, M. B. Rosson, G. Rothermel, M. Shaw, and S. Wiedenbeck, "The state of the art in end-user software engineering," *ACM Comput. Surv.*, vol. 43, no. 3, pp. 21:1–21:44, Apr. 2011.

[15] M. Burnett, J. Atwood, R. Walpole Djang, J. Reichwein, H. Gottfried, and S. Yang, "Forms/3: A first-order visual language to explore the boundaries of the spreadsheet paradigm," *J. Funct. Program.*, vol. 11, no. 2, pp. 155–206, Mar. 2001.

[16] S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer, "Wrangler: Interactive visual specification of data transformation scripts," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '11. New York, NY, USA: ACM, 2011, pp. 3363–3372.

[17] C. Kelleher and R. Pausch, "Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers," *ACM Comput. Surv.*, vol. 37, no. 2, pp. 83–137, Jun. 2005.

[18] G. Leshed, E. M. Haber, T. Matthews, and T. Lau, "Coscripter: Automating & sharing how-to knowledge in the enterprise," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '08. New York, NY, USA: ACM, 2008, pp. 1719–1728.

[19] L. Beckwith, C. Kissinger, M. Burnett, S. Wiedenbeck, J. Lawrance, A. Blackwell, and C. Cook, "Tinkering and gender in end-user programmers' debugging," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '06. New York, NY, USA: ACM, 2006, pp. 231–240.

[20] C. Bogart, M. Burnett, A. Cypher, and C. Scaffidi, "End-user programming in the wild: A field study of coscripter scripts," in *2008 IEEE Symposium on Visual Languages and Human-Centric Computing*, Sept 2008, pp. 39–46.

[21] B. Dorn and M. Guzdial, "Learning on the job: Characterizing the programming knowledge and learning strategies of web designers," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '10. New York, NY, USA: ACM, 2010, pp. 703–712.

[22] A. J. Ko, B. A. Myers, and H. H. Aung, "Six learning barriers in end-user programming systems," in *Proceedings of the 2004 IEEE Symposium on Visual Languages - Human Centric Computing*, ser. VLHCC '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 199–206.

[23] A. J. Ko and Y. Riche, "The role of conceptual knowledge in api usability," in *2011 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 2011, pp. 173–176.

[24] G. Wilson, "Software carpentry: Getting scientists to write better code by making them more productive," *Computing in Science Engineering*, vol. 8, no. 6, pp. 66–69, Nov 2006.

[25] B. Dorn and M. Guzdial, "Graphic designers who program as informal computer science learners," in *Proceedings of the Second International Workshop on Computing Education Research*, ser. ICER '06. New York, NY, USA: ACM, 2006, pp. 127–134.

[26] L. Ni and M. Guzdial, "Who am i?: Understanding high school computer science teachers' professional identity," in *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '12. New York, NY, USA: ACM, 2012, pp. 499–504.

[27] S. T. Phipps, L. C. Prieto, and E. N. Ndinguri, "Teaching an old dog new tricks: Investigating how age, ability, and self efficacy influence intentions to learn and learning among participants in adult education," *Academy of Educational Leadership Journal*, vol. 17, no. 1, p. 13, 2013.

[28] Z. Dornyei and I. Ottó, "Motivation in action: A process model of l2 motivation," 1998.

[29] M. B. Kery and B. A. Myers, "Exploring exploratory programming," in *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 2017, pp. 25–29.

[30] P. K. Chilana, C. Alcock, S. Dembla, A. Ho, A. Hurst, B. Armstrong, and P. J. Guo, "Perceptions of non-cs majors in intro programming: The rise of the conversational programmer," in *2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 2015, pp. 251–259.

[31] P. K. Chilana, R. Singh, and P. J. Guo, "Understanding conversational programmers: A perspective from the software industry," in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, ser. CHI '16. New York, NY, USA: ACM, 2016, pp. 1462–1472.

[32] A. Y. Wang, R. Mitts, P. J. Guo, and P. K. Chilana, "Mismatch of expectations: How modern learning resources fail conversational programmers," in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, ser. CHI '18. New York, NY, USA: ACM, 2018, pp. 511:1–511:13.

[33] "TensorFlow: An open source machine learning framework for everyone," https://www.tensorflow.org/, 2018, accessed: 2019-04-01.

[34] J. Corbin, "Basics of qualitative research," 1998.

[35] P. J. Guo, "Older adults learning computer programming: Motivations, frustrations, and design opportunities," in *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, ser. CHI '17. New York, NY, USA: ACM, 2017, pp. 7070–7083.

[36] ——, "Non-native english speakers learning computer programming: Barriers, desires, and design opportunities," in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, ser. CHI '18. ACM, 2018, pp. 396:1–396:14.

[37] "The Coding Train – YouTube channel," https://www.youtube.com/channel/UCvjgXvBlbQiydffZU7m1_aw, 2018, accessed: 2019-04-01.

[38] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of go without human knowledge," *Nature*, vol. 550, pp. 354–, Oct. 2017.

[39] A. Wigfield and J. S. Eccles, "Expectancy–value theory of achievement motivation," *Contemporary educational psychology*, vol. 25, no. 1, pp. 68–81, 2000.

[40] S. Kandel, A. Paepcke, J. M. Hellerstein, and J. Heer, "Enterprise data analysis and visualization: An interview study," *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, no. 12, pp. 2917–2926, Dec. 2012.

[41] K. Patel, N. Bancroft, S. M. Drucker, J. Fogarty, A. J. Ko, and J. Landay, "Gestalt: Integrated support for implementation and analysis in machine learning," in *Proceedings of the 23Nd Annual ACM Symposium on User Interface Software and Technology*, ser. UIST '10. New York, NY, USA: ACM, 2010, pp. 37–46.

[42] K. Wongsuphasawat, D. Smilkov, J. Wexler, J. Wilson, D. Mané, D. Fritz, D. Krishnan, F. B. Vigas, and M. Wattenberg, "Visualizing dataflow graphs of deep learning models in tensorflow," *IEEE Trans. Visualization & Comp. Graphics (Proc. VAST)*, 2018.

[43] M. Oquab, L. Bottou, I. Laptev, and J. Sivic, "Learning and transferring mid-level image representations using convolutional neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 1717–1724.

[44] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.

[45] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.

[46] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, and Y. Kafai, "Scratch: Programming for all," *Commun. ACM*, vol. 52, no. 11, pp. 60–67, Nov. 2009.

[47] M. Guzdial, "Learner-centered design of computing education: Research on computing for everyone," *Synthesis Lectures on Human-Centered Informatics*, vol. 8, no. 6, pp. 1–165, 2015.

[48] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436, 2015.

[49] C. Stephanidis, D. Akoumianakis, M. Sfyrakis, and A. Paramythis, "Universal accessibility in hci: Process-oriented design guidelines and tool requirements," in *Proceedings of the 4th ERCIM Workshop on User Interfaces for all, Stockholm, Sweden*, 1998, pp. 19–21.