Machine Programming: Turning Data into Programmer Productivity

Abdul Wasay
Intel Labs
abdul.wasay@intel.com

Nesime Tatbul
Intel Labs and MIT
tatbul@csail.mit.edu

Justin Gottschlich
Merly AI
justin.gottschlich@merly.ai

ABSTRACT

Machine programming is an emerging research area that improves the software development life cycle from design through deployment. We present a tutorial on machine programming research highlighting aspects relevant to the data systems community. We divide this tutorial into three parts: We begin with an introduction to machine programming introducing its three pillars: intention, invention, and adaptation. Then, we provide an overview of the data ecosystem central to all machine programming systems, highlighting challenges and novel opportunities relevant to the data systems community. Finally, we describe recent advances in machine programming research and how these directions use various data sets to improve the ease of creating and maintaining performant software systems.

PVLDB Reference Format:
Abdul Wasay, Nesime Tatbul, and Justin Gottschlich. Machine Programming: Turning Data into Programmer Productivity. PVLDB, 15(12): 3754-3757, 2022.
doi:10.14778/3554821.3554892

1 INTRODUCTION

Machine Programming. Programming (i.e., producing software systems) is a cognitively demanding task that requires extensive knowledge, experience, and a significant degree of creativity. This complexity is evident by the fact that technology companies spend a substantial portion of their operating cost to produce, maintain, and deploy software systems [18]. *Machine programming is an emerging area of research that develops automated tools and techniques to reduce this cost – cognitive, computational, and monetary – of producing and maintaining correct and efficient software systems while harnessing the full power of modern computing platforms* [14]. Due to its potential, machine programming is a fast-growing area of research with participation by several research groups across industrial and academic institutions [1, 6, 12, 14, 17, 19, 26, 30, 31, 33].

Tutorial Overview and Relevance. While machine programming research spans several communities, including software engineering, programming languages, machine learning, and user interface design, data is central to all machine programming systems. We present a 90-minute tutorial on machine programming research for a data systems audience, emphasizing avenues for cross-pollination between machine programming and data systems research.

(1) Machine Programming and its Three Pillars (20 min). We begin with an introduction to machine programming, positioning it with respect to other related areas such as program synthesis and automated programming. We also introduce the three pillars of machine programming – Intention, Invention, and Adaptation – to provide a frame of reference for the rest of the discussion.

(2) Data in Machine Programming Systems (35 min). Next, we provide an overview of the data ecosystem all machine programming research heavily relies on, such as code and telemetry data. First, we highlight the diversity and scale of data sources and formats within this ecosystem. Then, we emphasize how some of the core challenges in machine programming research directly connect to how machine programming systems collect, store, represent, and analyze diverse, multi-dimensional data sources. We then discuss open research challenges – data cleaning, integration, and representation – amenable to novel data systems research.

(3) Advances in Machine Programming Systems (35 min). Here, we examine emerging research directions in machine programming systems that facilitate the design, development, and deployment of various applications. While machine programming systems span a diverse array of applications, we pick the most relevant ones to a data systems audience. We first describe advances in code intelligence and automated debugging tools aimed at increasing the productivity of all programmers. Then, we describe advances in machine programming systems that support the life cycle of specific data-intensive applications, including cloud microservices, data science pipelines, and data flow applications.

Table 1 provides a representative (but not exhaustive) list of machine programming systems classified based on various dimensions relevant to this tutorial, including data sources, machine programming pillars, and application domains.

Audience. We design this tutorial for an audience with a data management background (students, academics, researchers, and industry practitioners). We will provide background knowledge on relevant topics from machine learning and programming languages.

Related Tutorials. Several related tutorials have focused on application of machine learning to automate various components of database systems [22, 24, 25, 39]. We, however, explore research aimed at improving programmer productivity in designing, developing, and deploying correct and performant computer systems. We consolidate research from various communities under this umbrella (i.e., machine programming) and highlight novel opportunities for data systems researchers.

Proceedings of the VLDB Endowment, Vol. 15, No. 12 ISSN 2150-8097.
doi:10.14778/3554821.3554892

Table 1: We classify machine programming systems across several dimensions including data sources and learning mechanisms.

MP System	Data Source(s)	Learning Type	MP Pillar(s)	Stage	Application Domain
CoPilot [5]	Code + documentation	Self-supervised	Intention + Invention	Development	General
ControlFlag [17]	Code repository	Self-supervised	Intention	Development	General
Ithemal [27]	Code + perf. profile	Supervised	Intention + Invention	Development	General
AutoPerf [1]	Telemetry data	Self-supervised	Adaptation	Development	General
Optdebug [16]	Provenance	Unsupervised	Intention + Adaptation	Debugging	DataFlow Systems
Dagger [33]	Provenance	Unsupervised	Adaptation	Debugging	DataFlow Systems
μqsim [40]	Telemetry data	Supervised	Adaptation	Debugging	Cloud Microservices
Sage [11]	Telemetry data	Unsupervised	Adaptation	Debugging	Cloud Microservices
Snorkel [31]	Knowledge Bases	Semi-supervised	Invention	Design	Data Science Pipelines
Holoclean [32]	Training data	Self-supervised	Invention	Debugging	Data Science Pipelines

2 MACHINE PROGRAMMING AND ITS THREE PILLARS

Machine Programming: What and Why. The pipeline that produces correct and efficient software has several stages, including design, development, debugging, optimization, deployment, and redesign. Machine programming research designs automated tools and techniques to improve the ease across all these stages of the software development pipeline. The eventual aim is to enable everyone to create correct and performant software systems [14]. Machine programming borrows techniques from machine learning, programming languages, and formal methods. Compared to other research directions such as program synthesis or automated programming that focus on a single stage, machine programming takes a holistic view of the software life cycle. This view enables us to explore a rich set of research directions in this tutorial.

Three Pillars. We introduce the three pillars of machine programming – Intention, Invention, and Adaptation – as a framework to think about existing and emerging systems.

(i) Intention. Those systems that make it easy for users to express what they want a software system to do without substantial programming effort fall under the Intention pillar. These include systems with visual and natural language interfaces, those that enable programming through examples, and ones with modular interfaces.

(ii) Invention. After a user specifies their intent, the pillar of Invention is concerned with systems that reduce the overhead of producing software systems that execute that intention correctly and as efficiently as possible. These include code generation and synthesis systems, auto-complete systems, and systems that catch correctness and performance bugs.

(iii) Adaptation. Finally, software systems do not operate in a vacuum and need to interact with various contexts such as application workloads, hardware configurations, and data sets. These context variables keep evolving, and software systems need to keep up to maintain correctness and efficiency. We classify systems that enable this flexibility under the Adaptation pillar. These include systems that help with performance regression testing, automated scaling, and management of deployed software.

3 DATA IN MACHINE PROGRAMMING

All machine programming systems heavily rely on data to uncover patterns, train models, and build formal logic. This data comes from various sources (e.g., code repositories, documentation, and telemetry data) and can have multiple formats (e.g., unstructured, textual, and time series).

Code Repositories. Public code repositories, such as Github, provide access to several billions of lines of code in hundreds of programming languages along with metadata such as documentation and revision history [38]. While these repositories contain a wealth of information to power machine programming systems, they suffer from two significant challenges: (i) They have a high proportion of duplicate projects, and (ii) it is increasingly hard to gauge the quality of code [2]. Recent research addresses these challenges by curating and cleaning data. For instance, CodeNet by IBM provides access to code snippets in 50 programming languages and various quality metrics [30]. Other similar sanitized data sets include POJ-104 and Google Code Jam [4, 37].

Benchmarks. Another active area of research is to establish benchmarks for the emerging use cases of machine programming. One popular benchmark is CodeXGLUE, which comes with data sets and machine learning models corresponding to several tasks such as code search, auto-complete, and translation. CodeXGLUE also hosts a leaderboard to rank various models for every task. DeathStarBench is another benchmark suite geared at cloud applications. It provides five microservices configurations that range from social media to drone coordination [12].

Incomplete and Synthetic Code + Natural Language. Online programming forums such as Stack Overflow and LeetCode are another rich source of data containing not just code but also other features like natural language explanation and peer ranking information. Recent evidence shows how data in various formats can help to improve the accuracy and scope of various machine programming tasks [28, 29].

Telemetry Data. Software and hardware telemetry data is becoming increasingly important for various machine learning systems. For instance, automatic performance regression systems (such as AutoPerf and OptDebug [1, 16]) or automatic cloud management systems (such as Sage and Sinan [11, 13]) make extensive use of

telemetry data for anomaly detection as well as root cause localization in deployed software systems.

Open Research Directions. These various sources of data introduce specific challenges, e.g., (i) how to clean and integrate multi-dimensional data from heterogeneous sources and formats while ensuring high data quality in an open-source and decentralized data ecosystem, (ii) what code representations to use for a given task and how to efficiently store and manage those representations, and (iii) there is the question of standardizing benchmarks for various machine programming tasks.

4 ADVANCES IN MACHINE PROGRAMMING SYSTEMS

General-Purpose MP Systems. First, we describe general-purpose machine programming research aimed at improving the life cycle of any software system.

(i) Easy Programmatic Interfaces. Research in this direction makes it easy for a user to specify their intention through intuitive interfaces without requiring substantial programming effort: Programming-by-example approaches can synthesize functions based on a few samples of their input-output behavior [8]. Plotcodder is a recent research system that automatically generates visualization code based on the data set to be visualized [7]. Additionally, various systems enable program synthesis and data exploration through touch-based, visual, and even augmented reality interfaces [20, 35].

(ii) Code Intelligence. This category of systems leverages artificial intelligence to make it easy to produce code by automating various tasks such as code search, code completion, and code-to-code translation [26]. In the code search domain, various systems map between natural language utterances and either domain-specific or general-purpose computer languages: Concode is a system that maps natural language to code within an existing program [21]. Code2Seq goes in the opposite direction generating natural language names for functions based on their code [3]. Similarly, there is work on predicting documentation from structured code [19]. In the code completion regime, Pythia is a state-of-the-art system that learns how to complete python code by learning directly from open-source repositories [36]. On the other hand, ControlFlag uses self-supervision to extract patterns within a code repository. ControlFlag can then use these patterns to flag incorrect patterns and suggest corrections [17]. Research in code search and code completion regime come together in Co-Pilot, a widely-used industrial system designed by Github [5]. Finally, recent advances in deep learning have enabled highly accurate code-to-code translation that in the past relied on hand-coded pattern matching [34].

(ii) Debugging. Complimentary to code intelligence research is the direction to automate the process of debugging an existing software system for both correctness and performance. ControlFlag and Hoppity are recent systems that employ deep learning approaches to debug correctness bugs through anomaly detection [10, 17]. In the performance debugging regime, Ithemal is a system that predicts the performance (in the form of CPU clock cycles) of code snippets without running them [27]. AutoPerf is another system in this area that helps localize performance bugs that may surface between any two versions of a deployed software system [1].

MP Systems for Data-intensive Applications. In this part, we focus on machine programming systems designed for data-intensive application areas, where data, in addition to code, determines correctness and performance.

(i) Data Flow Systems. Data Flow applications (such as those that use Hadoop, Map Reduce, or Spark) are a complex mix of both code and data. In data flow applications, a bug can be attributed to errors in data processing or program logic [16]. There is research to make it memory- and compute-efficient to track provenance in such applications [23]. Systems like Dagger, BigDebug, and TagSniff enable interactive debugging of this data provenance to localize bugs [9, 15, 23, 33]. Finally, Optdebug is a recent system that allows for an efficient combined data flow and logic debugging [16].

(ii) Cloud Microservices. The microservices model, where applications are deployed as loosely-coupled services, is increasingly being adopted by various applications, including social networks, web search, and drone coordination [12]. Different systems help to design and debug these complex applications. μqsim is one such system that uses statistics gathered from telemetry data to simulate the performance of a deployed microservice [40]. These models are used for testing and debugging purposes. Recent research combines profiling data sets with deep learning models to enable various tasks to improve debugging and deployment. For instance, Seer helps localize services or a set of services likely to result in a quality-of-service violation and suggests ways to remedy it [13]. Sage is another system that allows designers to generate alternative designs for a given microservice deployment and evaluate them [11].

(iii) Data Science Pipelines. Various research directions make it easy to create and repair data sets for data science and machine learning pipelines. For instance, Snorkel is one such system that automates the process of labeling data by allowing data scientists to specify labeling rules and then apply them to unlabeled data [31]. Holoclean is a system that builds probabilistic models of the data set, then uses these models to discover and correct errors (duplication or missing data) [32].

5 PRESENTERS

Abdul Wasay is a Research Scientist at *the Machine Programming Group* at Intel Labs. His research is at the intersection of systems and machine learning. He designs machine programming systems to enable better understanding of AI and cloud applications. His research appears in leading systems and machine learning venues such as SIGMOD, MLSys, and ICLR. He completed his Ph.D. in Computer Science from Harvard University. Wasay has also designed and taught several courses at Harvard University and Ashesi University (Ghana) to a diverse group of students.

Nesime Tatbul is a Senior Research Scientist at Intel Labs and MIT. She leads the Intel-MIT university collaboration program on data systems and artificial intelligence. Previously, she served on the CS faculty of ETH Zurich, after receiving a Ph.D. degree from Brown University. Her current research focuses on learned systems, time series analytics, and observability data management. She is one of the *co-authors of "The Three Pillars of Machine Programming"* [14], which provides the foundational vision behind this tutorial.

Justin Gottschlich is the Founder, CEO & Chief Scientist of Merly.ai, *a company aimed at automating software development using state-of-the-art machine programming systems.* Previously, Justin was a Principal AI Scientist and the *Founder & Director of Machine Programming Research* at Intel Labs. He co-founded the ACM SIGPLAN Machine Programming Symposium (MAPS) and serves as its Steering Committee Chair. He currently serves on the 2020 NSF Expeditions "Understanding the World Through Code". Justin received his PhD in Computer Engineering from the University of Colorado-Boulder in 2011 and has 40+ peer-reviewed publications, 50+ issued patents, with 100+ patents pending.

REFERENCES

[1] Mejbah Alam, Justin Gottschlich, Nesime Tatbul, Javier S Turek, Tim Mattson, and Abdullah Muzahid. 2019. A Zero-Positive Learning Approach for Diagnosing Software Performance Regressions. In *Advances in Neural Information Processing Systems*.

[2] Miltiadis Allamanis. 2019. The adverse effects of code duplication in machine learning models of code. In *Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*.

[3] Uri Alon, Omer Levy, and Eran Yahav. 2019. code2seq: Generating Sequences from Structured Representations of Code. In *International Conference on Learning Representations*.

[4] Tal Ben-Nun, Alice Shoshana Jakobovits, and Torsten Hoefler. 2018. Neural Code Comprehension: A Learnable Representation of Code Semantics. (2018).

[5] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* (2021).

[6] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* (2021).

[7] Xinyun Chen, Linyuan Gong, Alvin Cheung, and Dawn Song. 2021. PlotCoder: Hierarchical Decoding for Synthesizing Visualization Code in Programmatic Context. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics*.

[8] Xinyun Chen, Dawn Song, and Yuandong Tian. 2021. Latent Execution for Neural Program Synthesis. In *Advances in Neural Information Processing Systems*.

[9] Bertty Contreras-Rojas, Jorge-Arnulfo Quiané-Ruiz, Zoi Kaoudi, and Saravanan Thirumuruganathan. 2019. Tagsniff: Simplified big data debugging for dataflow jobs. In *Proceedings of the ACM Symposium on Cloud Computing*.

[10] Elizabeth Dinella, Hanjun Dai, Ziyang Li, Mayur Naik, Le Song, and Ke Wang. 2020. Hoppity: Learning Graph Transoformations to Detect and Fix Bugs in Programs. In *International Conference on Learning Representations*.

[11] Yu Gan, Mingyu Liang, Sundar Dev, David Lo, and Christina Delimitrou. 2021. Sage: practical and scalable ml-driven performance debugging in microservices. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*.

[12] Yu Gan, Yanqi Zhang, Dailun Cheng, Ankitha Shetty, Priyal Rathi, Nayan Katarki, Ariana Bruno, Justin Hu, Brian Ritchken, Brendon Jackson, et al. 2019. An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*.

[13] Yu Gan, Yanqi Zhang, Kelvin Hu, Dailun Cheng, Yuan He, Meghna Pancholi, and Christina Delimitrou. 2019. Seer: Leveraging big data to navigate the complexity of performance debugging in cloud microservices. In *Proceedings of the twenty-fourth international conference on architectural support for programming languages and operating systems*.

[14] Justin Gottschlich, Armando Solar-Lezama, Nesime Tatbul, Michael Carbin, Martin Rinard, Regina Barzilay, Saman Amarasinghe, Joshua B Tenenbaum, and Tim Mattson. 2018. The three pillars of machine programming. In *Proceedings of the 2nd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*.

[15] Muhammad Ali Gulzar, Matteo Interlandi, Seunghyun Yoo, Sai Deep Tetali, Tyson Condie, Todd Millstein, and Miryung Kim. 2016. Bigdebug: Debugging primitives for interactive big data processing in spark. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*.

[16] Muhammad Ali Gulzar and Miryung Kim. 2021. OptDebug: Fault-Inducing Operation Isolation for Dataflow Applications. In *Proceedings of the ACM Symposium on Cloud Computing*.

[17] Niranjan Hasabnis and Justin Gottschlich. 2021. ControlFlag: a self-supervised idiosyncratic pattern detection system for software control structures. In *Proceedings of the 5th ACM SIGPLAN International Symposium on Machine Programming*.

[18] Caryn Hubbard. [n.d.]. A Transparent Look Inside Our Process For Creating Buffer's Annual Budget. https://buffer.com/resources/creating-startup-budget/. (Accessed on 03/25/2022).

[19] Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. 2019. Codesearchnet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436* (2019).

[20] Stratos Idreos, Olga Papaemmanouil, and Surajit Chaudhuri. 2015. Overview of Data Exploration Techniques. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*.

[21] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. 2018. Mapping language to code in programmatic context. (2018).

[22] Alekh Jindal and Matteo Interlandi. 2021. Machine Learning for Cloud Data Systems: the Promise, the Progress, and the Path Forward. *Proc. VLDB Endow.* (2021).

[23] Avinash Kumar, Zuozhi Wang, Shengquan Ni, and Chen Li. 2020. Amber: a debuggable dataflow system based on the actor model. *Proceedings of the VLDB Endowment* (2020).

[24] Guoliang Li, Xuanhe Zhou, and Lei Cao. 2021. AI Meets Database: AI4DB and DB4AI. In *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*. ACM, 2859–2866.

[25] Guoliang Li, Xuanhe Zhou, and Lei Cao. 2021. Machine Learning for Databases. *Proc. VLDB Endow.* (2021).

[26] Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin B. Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, Ming Gong, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie Liu. 2021. CodeXGLUE: A Machine Learning Benchmark Dataset for Code Understanding and Generation. *CoRR* abs/2102.04664 (2021).

[27] Charith Mendis, Alex Renda, Saman Amarasinghe, and Michael Carbin. 2019. Ithemal: Accurate, portable and fast basic block throughput estimation using deep neural networks. In *International Conference on machine learning*.

[28] Alon Mishne, Sharon Shoham, and Eran Yahav. 2012. Typestate-based semantic code search over partial programs. In *Proceedings of the ACM international conference on Object oriented programming systems languages and applications*.

[29] Hila Peleg, Sharon Shoham, Eran Yahav, and Hongseok Yang. 2013. Symbolic automata for static specification mining. In *International Static Analysis Symposium*.

[30] Ruchir Puri, David S Kung, Geert Janssen, Wei Zhang, Giacomo Domeniconi, Vladimir Zolotov, Julian Dolby, Jie Chen, Mihir Choudhury, Lindsey Decker, et al. 2021. CodeNet: A Large-Scale AI for Code Dataset for Learning a Diversity of Coding Tasks. (2021).

[31] Alexander Ratner, Stephen H Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. 2020. Snorkel: Rapid training data creation with weak supervision. *The VLDB Journal* (2020).

[32] Theodoros Rekatsinas, Xu Chu, Ihab F Ilyas, and Christopher Ré. 2017. Holoclean: Holistic data repairs with probabilistic inference. *The VLDB Journal* (2017).

[33] El Kindi Rezig, Ashrita Brahmaroutu, Nesime Tatbul, Mourad Ouzzani, Nan Tang, Timothy Mattson, Samuel Madden, and Michael Stonebraker. 2020. Debugging large-scale data science pipelines using dagger. *Proceedings of the VLDB Endowment* (2020).

[34] Baptiste Roziere, Marie-Anne Lachaux, Lowik Chanussot, and Guillaume Lample. 2020. Unsupervised translation of programming languages. *Advances in Neural Information Processing Systems* (2020).

[35] Rishabh Singh and Armando Solar-Lezama. 2012. SPT: Storyboard Programming Tool. In *Computer Aided Verification - 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings*.

[36] Alexey Svyatkovskiy, Ying Zhao, Shengyu Fu, and Neel Sundaresan. 2019. Pythia: AI-assisted code completion system. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*.

[37] Farhan Ullah, Hamad Naeem, Sohail Jabbar, Shehzad Khalid, Muhammad Ahsan Latif, Fadi Al-Turjman, and Leonardo Mostarda. 2019. Cyber security threats detection in internet of things using deep learning approach. *IEEE Access* (2019).

[38] Martin Vechev, Eran Yahav, et al. 2016. Programming with "big code". *Foundations and Trends in Programming Languages* (2016).

[39] Abdul Wasay, Subarna Chatterjee, and Stratos Idreos. 2021. Deep Learning: Systems and Responsibility. In *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*.

[40] Yanqi Zhang, Yu Gan, and Christina Delimitrou. 2019. qSim: Enabling Accurate and Scalable Simulation for Interactive Microservices. In *Proceedings of the 2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)* (Madison, WI).