Research Statement

Eugene Wu, Columbia University
http://www.eugenewu.net/statement

I believe interactive data interfaces (e.g., visualizations, dashboards, spreadsheets), not traditional programming, will be the primary means to empower the next billion data users, and have the potential to be as ubiquitous as web pages and mobile applications are today. In fact, data interfaces are already indispensible in nearly every domain and every part of the data life-cycle, from data collection to cleaning, analysis, and decision making. However, the number and effectiveness of today's data interfaces are a mere fraction of what's possible. The major reason is that current tools do not address the unique challenges of designing data interfaces *and* ensuring that they are scalable and highly responsive. As a result, even interfaces with basic functionality require huge amounts of resources and expertise to build. To this end, the Data Visualization Management Systems (DVMS) project [43, 45] pursues three primary research directions to drastically simplify how interfaces are designed and created, and expand what interfaces are capable of.

The first direction develops systems to simplify interface development. I have developed novel abstractions to declaratively express and optimize data interfaces [43, 45]. For instance, I was the first to show that interactions are logically expressible as data lineage operations [30], which simplifies programming and benefits from lineage-based optimizations. I then built Smoke [30, 29, 31], the first lineage-supporting query engine to run interactive visualizations as fast as, or faster than, hand-optimized implementations. I have also developed the first tools to help designers make informed design decisions based on systems considerations and implications. These interface design tools [32] surface the interactivity achievable from the available system resources, and recommends optimizations needed to achieve the designer's desired level of responsiveness.

The second direction obviates the need for interface design and development. Even data scientists and programmers find it challenging to turn their analyses into usable interfaces, thus the Precision Interfaces system [57, 56, 55, 9] automatically generates interactive visualization interfaces from example analysis queries. This offers the potential to create analysis interfaces that are highly adapted to user needs, and create them at scale by monitoring data scientists' analyses or mining query logs generated by data systems.

The third direction expands interface capabilities from data presentation to explanation. When users encounter anomalous data in a visualization, they will want to understand why. In response, I first proposed the query explanation problem [47, 48] to generate predicates that describe input data errors that would "explain away" the anomalous outputs in the context of aggregation queries. I further extended the idea to training data debugging in machine learning (ML) analytic pipelines [53, 10, 21]. This user-centered approach towards data debugging [26] has led to significant followup work in the database community, and several companies, such as Honeycomb.io, have integrated these ideas into their analysis systems.

The emerging field of Human Data Interaction (HDI) studies how interfaces can best empower people to work with data. My approach is to identify fundamental HDI problems, and the above directions have just begun to tackle some of the core questions: what interfaces to build? how to build them? and what can they do? To do so, I combine data management and visualization techniques to develop new algorithmic and systems solutions, and demonstrate their effectiveness through careful evaluation and user studies. This work, with students, postdocs, and collaborators, has led to 16 full papers at top conferences (5xVLDB, 5xSIGMOD, 1xCIDR, 1xVis, 1xTVCG, 2xICWSM, 1xSOCC), and 53 total papers since starting at Columbia. My students and postdocs have gone to Microsoft Research and Google Research, and won the Google PhD Fellowship and the 2020 SIGMOD Student Research Competition. I have been supported and recognized by 4 NSF grants and 2 Google and 2 Amazon awards, adoption by multiple companies, 2 best-of-conference citations, a 2015 SIGMOD best demo award, the 2018 VLDB Test-of-Time award, and the NSF CAREER. I have made early contributions to crowdsourced databases [22, 23, 12], complex event processing [44, 11], mining web tables [6, 7], and application-sensitive data cleaning [20, 18, 17, 38, 39], and neural network debugging [8, 37, 36].

1 Systems to Simplify Interface Development

Designing and building interactive data interfaces is a highly iterative process to identify the best visual data representations and interactions for a given analysis task under challenging user- and systems-level constraints. Users expect immediate responses irrespective of the analysis complexity and data size, while large data volumes necessitate complex systems that span browser, server, database, and cloud platforms. Implementing these layered systems requires piecing together, and optimizing across, different technologies. This is difficult for even professional developers, and completely out of reach for data scientists and users. Further, every interface design change requires manual re-implementation and re-optimization, which grinds the iterative design process to a halt. New abstractions and development tools are needed to support rapid interface and system co-design.

1.1 Declarative Abstractions

The enduring success of database systems is thanks to its declarative abstraction: given a query, the database automatically optimizes and executes the query. Although data interfaces also generate queries in response to user interactions, optimizing individual queries is insufficient to ensure user-facing responsiveness [14]. Similarly, declarative visualization libraries focus on presentation and layout, do not address data processing bottlenecks. My goal is to create a declarative language that is compatible with existing visualization and design libraries, and can express the data flows that underlie interface interactions in a form that is amenable to optimization. While we have developed several candidate languages [43, 45, 32, 56], I have created two systems that provide key abstractions: Smoke, which uses lineage to declaratively express interactions, and Khameleon, which abstracts away communication bottlenecks that are endemic in networked applications.

Lineage-based Interactions: I discovered a connection between interaction and data lineage [30] that led to simpler development abstractions, novel optimization opportunities, and functionality that can benefit a wide variety of interactive applications: interfaces render data as pixels on the screen, and when the user points to or manipulates pixels, they are implicitly manipulating the underlying data (the data lineage) of those pixels. If efficient lineage tracking were possible, then it could serve as the basis to express many useful interactions. In addition, that fast interactive visualizations exist is proof that it is possible. For this reason, we developed Smoke, the first practical lineage tracking techniques that reduced the overhead from $\geq 1000\times$ in prior work [49, 42, 15, 5] to $\approx 0.8\times$ [31]. The key insight is that lineage tracking can "piggyback" on top of query execution and eliminate redundant work. Smoke is fast enough to declaratively express interactive visualizations and data profilers using lineage constructs, *and run as fast as, or faster than*, state-of-the-art and hand-tuned implementations [31, 29]. Lineage-supporting systems have the potential to support novel interaction functionalities such as data explanation, cross-application linking and interactions, interaction histories, and interaction-by-example [29], and we are integrating ideas from Smoke in an in-memory columnar engine to show these functionalities in practice.

Communication: Communication is fundamental in any networked interactive application, however masking communication delays is a complex art that depends on the application needs and demands rare networking and interaction design expertise. The classic approach to mask communication latencies trades extra bandwidth for lower perceived latency by predicting future requests and prefetching them. However, bandwidth is often a bottleneck for interactive interfaces: a single gesture can easily generate dozens or hundreds of requests that return large, data-dense responses. The requests already overwhelm the network, and prefetching exacerbates the issue. Although it is possible for some developers to create one-off solutions to prioritize user-level responsiveness, each application has different priorities, types of requests, and network settings.

Khameleon [24] is a general prefetching framework for interactive applications that lets developers focus

on policy decisions rather than low level implementation and optimization details. Rather than solely focus on improving the predictor, we use a combination of server-push and progressively encoded responses to pose prefetch as a novel scheduling problem. The server continuously pushes fragments of response data to the client, where a fragment is sufficient to render an approximate, lower quality response. The scheduling formulation lets the developer focus on application-specific policy decisions to balance response quality (by dedicating all bandwidth to a few likely requests) and user-latency (by sending a tiny amount of many potential requests). By carefully managing network utilization, Khameleon reduces response latency by 2 orders of magnitude from >50 sec to ≤ 30 millisec on real AT&T LTE network traces, while it improving rendering quality, as compared to client-based prefetching that has access to an oracle prediction model. We show that porting state-of-the-art visualization-specific prefetch techniques [25] is simple (≈ 50 lines of code), and makes it easy to tune policies that further reduce latency by up to $8\times$.

Additional Opportunities: This declarative perspective helped me define and pursue additional novel opportunities. For example, instead of semantic query equivalence, I proposed "perceptual equivalence" [50] to describe approximate visualization results that are perceptually indistinguishable from the full results. I also developed perceptual models [27, 35, 46] to quantify this effect, and the first query optimizations [4, 28] to leverage perceptual models. As another example, client interfaces make asynchronous requests to avoid blocking the user interface (UI), but this easily leads to UI inconsistencies; I helped define the first approaches that borrow database concurrency control and provenance techniques to manage asynchrony in visualizations at the design and programming levels [54, 52, 51, 45].

1.2 Physical Visualization Design

Interface design is an iterative process that requires close coordination between the designer and developer in order to meet latency expectations: small interface changes may require major system changes, while implementation and optimization decisions introduce "performance cliffs" that the designer must avoid. Unfortunately, design and systems implementation are very different skills, and often necessitate multiple teams that are difficult to coordinate. Yet, no tools exist to aid this design process.

I envisioned the first interface design tool, akin to Photoshop, that facilitates rapid interface and system co-design [32]. The tool is initialized with existing system constraints and database information. As the designer creates the interface and adds interactions, the tool alerts the designer when latency expectations for certain interactions cannot be met, suggests system optimizations, and simulates their latency implications within the interface. In addition, it helps forecast additional resource costs as the database grows. To support this, I proposed Physical Visualization Design (PVD). Similar to physical database design tools that recommend indexes and views to accelerate query workloads, PVD takes as input an interface specification, an existing database, and a library of optimization techniques, and recommends system architectures—data structures to create, client-server data placement policies—to meet the designer's latency expectations within resource limits. Our 2020 demonstration showed a drag-and-drop interface design tool that recommends indexes and data placement policies; our full submission later this year supports real-world interfaces and optimizations.

2 Automatic Interface Generation

Data analysis interfaces restrict their expressive power in exchange for usability and simplicity. Similarly, interface design tools and dashboard builders also restrict their expressive power in exchange for speeding up data interface development and deployment. For instance, existing tools restrict the analyses to those expressible by e.g., parameterized queries, data cube operations, group-by aggregations, or single table queries. Unfortunately, data analyses do not fit into pre-canned templates and require the full expressive power of SQL,

which makes designing and building interfaces for the analyses considerably more challenging. The Precision Interfaces (PI) project [57, 56, 55, 9] is the first to study whether it is possible to *automatically generate* interfaces from these analysis queries. If possible, custom interfaces could be created by simply performing the intended analysis, by monitoring live analysis sessions, or by mining existing query logs (e.g., collected by existing data systems) to synthesize shared interfaces. Further, these interfaces would be highly adapted to individual users' analysis workflows, reduce cognitive load and user errors, and improve data accessibility.

My key insight is that interactive interfaces are not arbitrary programs. Instead, interactions change an underlying program in systematic ways: e.g., a slider controls a numeric parameter, a button replaces a query. Thus, an interface expresses the set of queries needed for the desired analysis task, and the input queries are a sample sequence generated from this "latent" interface. This project has developed a formal model of the mapping problem from queries to an interactive interface [57], and used it to generate interactive widgets [57] for different interaction modalities [55] (e.g., natural language, touch), and determine layouts based on screen size [9]. The work so far has solely focused on generating *interactions* to express the input queries, and the current work is to generate full interactive interfaces that support data visualizations and account for interface layout. This is difficult because the semantics of a visualization's interactions changes depending on the visualization type and specification, and because quantifying interface "goodness" remains an open problem. Our early findings show that many existing interfaces can be automatically generated solely by providing a handful of queries that the interfaces produce.

3 From Presentation to Explanation

Today's data interfaces focus on data presentation, and primarily provide interactions for low-level tasks such as filtering, grouping, panning, and zooming. However, users don't simply want to see a visualization—they want to understand what they are seeing and ask questions about it. This motivated me to define the *query explanation* problem [47, 48]. Given a user "complaint" that query results rendered in a visualization are surprising (e.g., why is the average temperature so high?), the system proposes hypotheses to "explain-away" these complaints (e.g., ignoring sensor 18 would have kept temperatures stable). These explanations describe predicate-based interventions that, if applied to the database, would help address the user complaints. This work has led to a number of followups in academia [3, 33, 34] and industry [2, 1].

A key insight is that explanation has strong ties to data cleaning. For instance, when explaining an anomaly—"*if we ignored this data and fixed these values, then the output would look normal*"—each step (e.g., ignore, fix) is also a cleaning intervention. However, cleaning typically focuses on a given dataset, whereas explanations take the downstream analysis into account. This novel complaint-driven data debugging is more user-friendly and helps identify data errors that matter to the use case. I have applied this concept in the first works that clean data to improve ML model [20, 16, 18, 17, 19], identify past erroneous transactions [41, 40], and interactively identify data errors [13]. The latter is in collaboration with The Earth Institute's Financial Instruments Sector Team to aid national farmer drought protection programs in Ethiopia and Zambia.

I recently proposed complaint-driven data debugging for ML analysis workflows [26]. Training data is currently debugged by writing rules, detecting syntactic errors, or relying on labeled prediction errors. In contrast, evidence of data errors is often only detected in the output of downstream analytics or by consumers of the models and predictions. There is a need to translate anomalies found downstream into errors in the data sources. To this end, I developed the first whitebox approach [53, 10] that combines influence analysis with query explanation to identify training record interventions for queries that use ML, as well as blackbox approaches that support general data science pipelines by leveraging Bayesian hyper-parameter optimization [21]. ML workflows often span multiple teams, and this approach helps combine expertise local to each team to better pinpoint data errors and improve debugging productivity.

References

[1] Diving into data with honeycomb drilldown.

[2] Sisu: Accelerate data exploration.

[3] F. Abuzaid, P. Kraft, S. Suri, E. Gan, E. Xu, A. Shenoy, A. Ananthanarayan, J. Sheu, E. Meijer, X. Wu, J. F. Naughton, P. Bailis, and M. Zaharia. Diff: a relational interface for large-scale data explanation. *The Vldb Journal*, pages 1–26, 2020.

[4] D. Alabi and E. Wu. Pfunk-h: Approximate query processing using perceptual models. *HILDA*, 2016.

[5] B. S. Arab, S. Feng, B. Glavic, S. Lee, X. Niu, and Q. Zeng. Gprom - a swiss army knife for your provenance needs. *IEEE Data Eng. Bull.*, 41:51–62, 2018.

[6] M. Cafarella, A. Halevy, D. Z. Wang, H. Lee, J. Madhavan, C. Yu, and E. Wu. Ten years of web tables. *PVLDB Invited Paper*, 2018.

[7] M. J. Cafarella, A. Y. Halevy, Y. Zhang, D. Z. Wang, and E. Wu. Uncovering the relational web. In *WebDB*, 2008.

[8] Y. Chen, Y. Shi, B. Chen, T. Sellam, C. Vondrick, and E. Wu. Deep neural inspection using deepbase. *NeurIPS LearnSys Workshop*, 2018.

[9] Y. Chen and E. Wu. Monte carlo tree search for generating interactive data analysis interfaces. In *Intelligent Process Automation (IPA)*, 2020.

[10] L. Flokas, Y. Wu, J. Wang, and E. Wu. Towards complaint-driven ml workflow debugging. In *MLOps*, 2020.

[11] D. Gyllstrom, E. Wu, H.-J. Chae, Y. Diao, P. Stahlberg, and G. Anderson. Sase: Complex event processing over streams (demo). In *CIDR*, 2007.

[12] D. Haas, J. Wang, E. Wu, and M. J. Franklin. Clamshell: Speeding up crowds for low-latency data labeling. *PVLDB*, 9:372–383, 2015.

[13] Z. Huang and E. Wu. Reptile: Aggregation-level explanations for hierarchical data. In *In Progress*, 2021 (anticipated).

[14] L. Jiang, P. Rahman, and A. Nandi. Evaluating interactive data systems: Workloads, metrics, and guidelines. *Proceedings of the 2018 International Conference on Management of Data*, 2018.

[15] G. Karvounarakis, Z. G. Ives, and V. Tannen. Querying data provenance. In *SIGMOD Conference*, 2010.

[16] S. Krishnan, M. Franklin, K. Goldberg, and E. Wu. Boostclean: Automated error detection and repair for machine learning. In *Tech Report*, 2017.

[17] S. Krishnan, M. J. Franklin, K. Goldberg, J. Wang, and E. Wu. Activeclean: An interactive data cleaning framework for modern machine learning. In *SIGMOD (demo)*, 2016.

[18] S. Krishnan, J. Wang, E. Wu, M. J. Franklin, and K. Goldberg. Activeclean: interactive data cleaning for statistical modeling. *PVLDB*, 2016.

[19] S. Krishnan and E. Wu. Palm: Machine learning explanations for iterative debugging. In *HILDA*, 2017.

[20] S. Krishnan and E. Wu. Alphaclean: Automatic generation of data cleaning pipelines. *arXiv*, 2018.

[21] B. Lockhard, J. Wang, and E. Wu. Explaining sql-ml queries with bayesian optimization. In *ArXiv*, 2021.

[22] A. Marcus, E. Wu, D. R. Karger, S. R. Madden, and R. C. Miller. Crowdsourced databases: Query processing with people. In *CIDR*, 2011.

[23] A. Marcus, E. Wu, D. R. Karger, S. R. Madden, and R. C. Miller. Human-powered sorts and joins. In *VLDB*, 2011.

[24] H. Mohammed, Z. Wei, R. Netravali, and E. Wu. Continuous prefetch for interactive data applications. In *VLDB*, 2020.

[25] D. Moritz, B. Howe, and J. Heer. Falcon: Balancing interactive latency and resolution sensitivity for scalable linked visualizations. *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, 2019.

[26] F. Neutatz, B. Chen, Z. Abedjan, and E. Wu. From cleaning before ml to cleaning for ml. In *IEEE Data Engineering Bulletin*, 2021.

[27] M. Procopio, A. Mosca, C. Scheidegger, E. Wu, and R. Chang. Impact of cognitive biases on progressive visualization. In *TVCG*, 2021.

[28] M. Procopio, C. Scheidegger, E. Wu, and R. Chang. Load-n-go: Fast approximate join visualizations that improve over time. In *DSIA*, 2017.

[29] F. Psallidas and E. Wu. Demonstration of smoke: A deep breath of data-intensive lineage applications. In *SIGMOD (demo)*, 2018.

[30] F. Psallidas and E. Wu. Provenance in interactive visualizations. In *HILDA*, 2018.

[31] F. Psallidas and E. Wu. Smoke: Fine-grained lineage at interactive speeds. In *VLDB*, 2018.

[32] L. Ramjit, S. Mitra, R. Netravali, and E. Wu. Physical visualization design. In *VLDB*, 2021.

[33] S. Roy, L. Orr, and D. Suciu. Explaining query answers with explanation-ready databases. *Proc. VLDB Endow.*, 9:348–359, 2015.

[34] S. Roy and D. Suciu. A formal approach to finding explanations for database queries. *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, 2014.

[35] G. Ryan, A. Mosca, R. Chang, and E. Wu. At a glance: Approximate entropy as a measure of line chart visualization complexity. In *InfoVIS*, 2018.

[36] T. Sellam, K. Lin, I. Huang, C. Vondrick, and E. Wu. "i like the way you think!" inspecting the internal logic of recurrent neural networks. In *SysML*, 2018.

[37] T. Sellam, K. Lin, I. Y. Huang, M. Yang, C. Vondrick, and E. Wu. Deepbase: Deep inspection of neural networks. *SIGMOD*, 2019.

[38] P. Wang, Y. He, R. Shea, J. Wang, and E. Wu. Deeper: A data enrichment system powered by deep web. In *SIGMOD (demo)*, 2018.

[39] P. Wang, R. Shea, J. Wang, and E. Wu. Progressive deep web crawling through keyword qeries for data enrichment. In *SIGMOD*, 2019.

[40] X. Wang, A. Meliou, and E. Wu. Qfix: Demonstrating error diagnosis in query histories. *SIGMOD (demo)*, 2016.

[41] X. Wang, A. Meliou, and E. Wu. Qfix: Diagnosing errors through query histories. *SIGMOD*, 2017.

[42] J. Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *CIDR*, 2005.

[43] E. Wu, L. Battle, and S. R. Madden. The case for data visualization management systems: Vision paper. *PVLDB*, 2014.

[44] E. Wu, Y. Diao, and S. Rizvi. High-performance complex event processing over streams. In *SIGMOD*. ACM, 2006.

[45] E. Wu, Z. M. Fotis Psallidas, H. Zhang, L. Rettig, Y. Wu, and T. Sellam. Combining design and performance in a data visualization management system. In *CIDR*, 2017.

[46] E. Wu, L. Jiang, L. Xu, and A. Nandi. Graphical perception in animated bar charts. *arXiv*, 2016.

[47] E. Wu and S. Madden. Scorpion: Explaining away outliers in aggregate queries. In *VLDB*, 2013.

[48] E. Wu, S. Madden, and M. Stonebraker. A demonstration of dbwipes: clean as you query. *PVLDB (demo)*, 2012.

[49] E. Wu, S. Madden, and M. Stonebraker. Subzero: a fine-grained lineage system for scientific databases. In *ICDE*, 2013.

[50] E. Wu and A. Nandi. Towards perception-aware interactive data visualization systems. In *DSIA at VIS*, 2015.

[51] Y. Wu, R. Chang, J. Hellerstein, A. Satyanarayan, and E. Wu. Diel: Interactive visualization beyond the here and now. In *ArXiV*.

[52] Y. Wu, R. Chang, J. Hellerstein, and E. Wu. Facilitating exploration with interaction snapshots under high latency. In *InfoVIS (short paper)*, 2020.

[53] Y. Wu, L. Flokas, J. Wang, and E. Wu. Complaint-driven training data debugging for query 2.0. In *SIGMOD*, 2020.

[54] Y. Wu, J. M. Hellerstein, and E. Wu. A devil-ish approach to inconsistency in interactive visualizations. *HILDA*, 2016.

[55] H. Zhang, V. Rai, T. Sellam, and E. Wu. Precision interfaces for different modalities. In *SIGMOD (demo)*, 2018.

[56] H. Zhang, T. Sellam, and E. Wu. Precision interfaces. In *HILDA*, 2017.

[57] H. Zhang, T. Sellam, and E. Wu. Mining precision interfaces from query logs. In *SIGMOD*, 2019.