Talaria: Interactively Optimizing Machine Learning Models for Efficient Inference

Fred Hohman
Apple
Seattle, WA, USA
fredhohman@apple.com

Chaoqun Wang
Apple
Beijing, China
chaoqun_wang@apple.com

Jinmook Lee
Apple
Cupertino, CA, USA
jinmook_lee@apple.com

Jochen Görtler*
Independent Researcher
Walldorf, Germany
me@jgoertler.com

Dominik Moritz
Apple
Pittsburgh, PA, USA
domoritz@apple.com

Jeffrey P. Bigham
Apple
Pittsburgh, PA, USA
jbigham@apple.com

Zhile Ren
Apple
Seattle, WA, USA
zhile_ren@apple.com

Cecile Foret
Apple
Cupertino, CA, USA
cforet@apple.com

Qi Shan
Apple
Seattle, WA, USA
qshan@apple.com

Xiaoyi Zhang
Apple
Seattle, WA, USA
xiaoyiz@apple.com

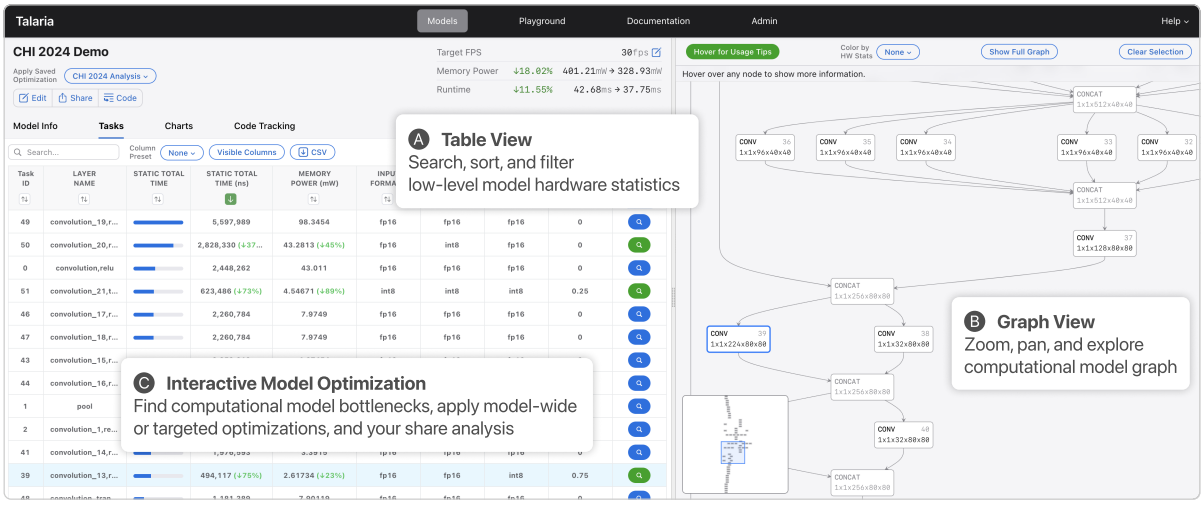

Figure 1: Talaria enables ML practitioners to compile models to hardware, jointly visualize their operations in the (A) Table View and (B) Graph View, while simulating a suite of (C) Interactive Model Optimization options to improve hardware inference efficiency. In this example, a user has sorted the operations by their compute time, selected one (highlighted in blue in both the table and graph), and applied an optimization that saves 18.02% memory power and 11.55% runtime latency.

ABSTRACT

On-device machine learning (ML) moves computation from the cloud to personal devices, protecting user privacy and enabling intelligent user experiences. However, fitting models on devices with limited resources presents a major technical challenge: practitioners need to optimize models and balance hardware metrics such as model size, latency, and power. To help practitioners create *efficient ML models*, we designed and developed Talaria: a model visualization and optimization system. Talaria enables practitioners to compile models to hardware, interactively visualize model statistics, and simulate optimizations to test the impact on inference metrics. Since its internal deployment two years ago, we have evaluated Talaria using three methodologies: (1) a log analysis highlighting its growth of 800+ practitioners submitting 3,600+ models; (2) a usability survey with 26 users assessing the utility of 20 Talaria features; and (3) a qualitative interview with the 7 most active users about their experience using Talaria.

*Work done at Apple.

CHI '24, May 11–16, 2024, Honolulu, HI, USA

ACM ISBN 979-8-4007-0330-0/24/05
https://doi.org/10.1145/3613904.3642628

CCS CONCEPTS

• **Human-centered computing** → **Visualization systems and tools**; **Interactive systems and tools**; • **Computing methodologies** → *Machine learning*; *Artificial intelligence.*

KEYWORDS

Efficient machine learning, model compression, on-device machine learning, interactive systems, visual analytics

ACM Reference Format:
Fred Hohman, Chaoqun Wang, Jinmook Lee, Jochen Görtler, Dominik Moritz, Jeffrey P. Bigham, Zhile Ren, Cecile Foret, Qi Shan, and Xiaoyi Zhang. 2024. Talaria: Interactively Optimizing Machine Learning Models for Efficient Inference. In *Proceedings of the CHI Conference on Human Factors in Computing Systems (CHI '24), May 11–16, 2024, Honolulu, HI, USA.* ACM, New York, NY, USA, 19 pages. https://doi.org/10.1145/3613904.3642628

1 INTRODUCTION

A continuing trend within machine learning (ML) research and development is to move inference computation away from cloud servers and instead on to personal computing [4–6] and edge devices [54]. Commonly referred to as on-device ML [32], or colloquially tinyML [89], this approach: (1) protects user privacy since data does not leave a user's device when computing inference, (2) enables new user experiences, especially for applications with strict latency requirements (e.g., inference at high refresh rates), (3) supports more portable experiences since models do not require internet access, and (4) allows developers without extensive compute resources to deliver ML experiences, reducing cost and the environmental impact of large servers. However, as the latest ML models continue to grow in size (e.g., neural networks with hundreds of billions of parameters [28, 78, 88, 100]), creating *efficient ML models* that can run inference on resource-constrained devices, such as phones, tablets, or wearables, is challenging, as deployment requires practitioners to optimize and compress their models while maintaining acceptable accuracy [86].

Besides model quality metrics (e.g., accuracy), how do ML practitioners effectively optimize and balance on-device inference efficiency, such as model size, power, and latency [8, 42]? Efficient ML research and development is still nascent, and the state-of-the-art is rapidly changing [24, 35, 76, 96, 99]. Best practices are largely undocumented or still forming [61, 89]. Much of the progress in efficient ML focuses on contributing novel compression algorithms—unfortunately much less work focuses on developing practical tools to help people successfully apply and understand the benefits of compression. As efficient ML techniques are driven forward by advances in hardware engineering and ML research, there remains a major barrier in helping ML practitioners apply these techniques for designing real-world and intelligent ML user experiences.

The tooling for developing efficient ML models is underexplored, underdeveloped, yet rich with opportunity [42]. In this timely area, better tools can have an outsized impact. Tooling for ML is often a force multiplier, enabling practitioners of varying expertise to develop models on their own. Interactive tools for model optimization and compression is a new direction of research, where the few existing works only scratch the surface. Beyond communicating the effect of applying specific algorithmic compression techniques [25, 53, 94], there are many other components of efficient ML development where interactive visualization could help practitioners create ML-powered, on-device user experiences.

To help ML practitioners build efficient models, we designed and developed Talaria: a model optimization and visualization system, informed by and built with expert ML practitioners at Apple that specialize in developing efficient models on-device. Talaria compiles models to hardware, and visualizes low-level hardware and model statistics through a split interface showing an interactive table and model graph, as shown in Figure 1. Talaria also simulates a suite of model optimizations to instantly show the impact on a model's inference efficiency (e.g., latency and memory). ML practitioners can apply these optimizations at the model level, or at the individual hardware operation level. The system is model agnostic and supports models for arbitrary ML tasks, such as vision (e.g., classification, object detection, segmentation), natural language processing, and sensing applications.

As the field of efficient ML matures, we expect model evaluation tooling to support practitioners in optimizing their models over both model behavioral metrics (e.g., accuracy, precision, recall) as well as hardware specific metrics (e.g., model size, latency, power consumption). However, everything comes at a cost, and in ML, the CACE principle [75], *"Changing Anything Changes Everything,"* continues to hold. Shrinking a model to reduce its size, latency, and power, while maintaining its accuracy and quality is extremely challenging in practice [42]. In this work, we intentionally focus on the new and novel challenges brought by moving ML inference onto personal computing devices for enabling user experiences powered by ML. Therefore, Talaria is scoped to help practitioners address evaluating a model's hardware metrics under the task of *on-device inference* (further discussed in Section 2.3).

We developed Talaria over 2 years, and report on 3 evaluations. First, we present a log analysis showing Talaria's successful adoption within our organization. Next, we discuss the results from a usability survey with 26 ML practitioners where they rate the utility of 20 different system features. Lastly, we detail the results from qualitative interviews with the 7 most active users to learn about their experience using Talaria and what improvements could be made to better help them create efficient models.

Our contributions include:

- **Formative research with 12 ML practitioners on model optimization.** Through a needfinding survey and participatory design sessions with low-fidelity prototyping, we outline the challenges and tasks of optimizing a model's power consumption, memory footprint, and inference latency in order to create efficient ML models.
- **Talaria: an interactive visualization system for creating efficient ML models.** Talaria compiles models to hardware, visualizes their low-level statistics and computational graph together, while simulating multiple model optimizations for testing inference efficiency (e.g., latency and memory). The web-based system allows users to interact with large models (e.g., thousands of operations) in real time. Talaria also introduces a mechanism to map hardware operations back to a model's source code. Lastly, the system supports collaborative model optimization by letting users

save optimizations and send a single URL to their colleagues to fork and continue their work.
- **Findings from three evaluations of Talaria deployed within ML research and development teams.** We conduct a log analysis to inspect the adoption of our system over time (800+ unique users uploaded over 3,600 models), a usability survey with 26 ML practitioners to rate and assess the utility of 20 system features, and a semi-structured qualitative interview with the 7 most active users to learn about their experience using Talaria for model optimization.

We believe efficient ML, specifically for on-device use cases, is a rich and untapped area of AI/ML for the human-computer interaction community to engage with. There is a large gap between current tools today and what practitioners need. We hope our work emphasizes the need and importance of tooling for optimizing models, and inspires future interdisciplinary work on interactive interfaces for creating intelligent and efficient ML user experiences.

2 BACKGROUND AND RELATED WORK

2.1 Model Compression Techniques

To shrink models, efficient ML practitioners use a variety of strategies, from principled architecture decisions to ad-hoc tricks-of-the-trade. One class of techniques is model compression: optimizations to various components of a model to minimize the amount of computational resources it needs. Categories of compression techniques (illustrated in Figure 2) include quantization [27], palettization [18, 93], pruning (i.e., sparsification [27, 40]), and other modeling specific techniques (e.g., distillation [27, 33, 65], efficient neural architectures [20, 45, 73, 81, 87, 98], and dynamic architectures [102]). Each technique is truly a family of techniques, with many nuanced variations that can be also combined together [37]. The following surveys detail compression techniques: [17, 20, 23, 57].

In this work, the compression techniques we use are quantization (Figure 2A), pruning (Figure 2B), and palettization (Figure 2C). For brief context, *quantization* converts the inputs, outputs, and/or weights of a model from high-precision formats (e.g., `fp32`) to lower-precision formats (e.g., `fp16`, `int8`, and even `int2`) [27]. *Weight pruning* removes the least-important parameters (e.g., weights, bias) of a model to make it smaller. The motivation is that modern neural networks are overparameterized, such that removing parameters will minimally impact the final prediction [27, 40]. Lastly, *palettization* maps the weights of a model to a discrete set of precomputed (or learned) values. Inspired by an artist's "palette," the idea is to map many similar values to one average or approximate value, then use those new values for computing inference. While there are many types of compression techniques, we focus on these three due to their popularity, performance, and common use.

2.2 Existing Compression Resources

Since investing in model compression is typically only needed for applications where models will run on-device, research and best practices for ML optimization is much more limited compared to ML in general. While surveys detail different compression techniques [17, 20, 23], most existing practical guidance stems from online tutorials and documentation from popular ML libraries. Examples include TensorFlow's model optimization toolkit and blog post on quantization-aware training [82, 83]; PyTorch's experimental support for quantization [66], sparsity [67], and it's accompanying examples [68]; Google's quantization extension to Keras called QKeras [31]; Microsoft's Neural Network Intelligence package and tool [58]; Intel's Neural Compressor library [47]; and Apple's MLX framework [38] and DNIKit [90]. For targeting specific hardware, other examples focus on speeding up inference on FPGAs [26] and compressing Core ML models to run on Apple platforms [7]. Lastly, the appropriately named TinyML community has emerged around this topic, which published a book [89] on developing models for always-on, low-power use cases.

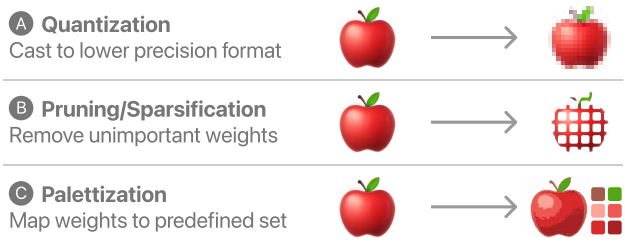

Figure 2: An illustration of three common model compression techniques built into Talaria. (A) Quantization converts data types from high-precision formats (e.g., `fp32`) to low-precision formats (e.g., `int8`). (B) Pruning/Sparsification removes unnecessary weights from neural networks. (C) Palettization maps model weights to a discrete set of precomputed (or learned) values.

2.3 On-device Inference v. On-device Training

It is important to clarify a distinction between on-device *inference* and on-device *training*. In our work, we focus on the more commonly studied and applied component of on-device inference: computing a prediction from a pretrained model loaded on a device with limited compute resources, such as a phone, tablet, or wearable, that have smaller memory and power capacity [42]. On these specific mobile computing devices, it is rare to train a model from scratch. In some ML contexts where personalization is needed, perhaps a model requires fine-tuning on a user's data on-device; however, this scenario is much less common than training a model offline and deploying it onto a mobile device to run inference [42]. While on-device inference and training share many similar challenges, and both could benefit from interactive tools and visualization, training on-device models is not as commonplace and usually requires more resources [101]. Thus, we intentionally scope our work on building tools for model optimization for ML that will run on-device inference. For resources on the current research and challenges around on-device learning instead, see the following surveys: [24, 55, 60, 101].

2.4 Visualization for Model Evaluation

Since the boom of ML innovation over a decade ago, there have been many visual analytics systems designed for most stages of the ML development cycle. This hybrid research direction of combining visualization and ML has made significant contributions

to model evaluation [41]. For different modeling tasks, tools for visualizing metrics (e.g., accuracy, precision, recall) on subsets of data [1, 15, 16, 91] and tools for exploring large ML datasets [10, 46] help practitioners compare and evaluate how well ML models generalize to unseen data. Example ML tasks incorporating visualization include data classification [3, 36, 70], image classification [19], object detection [34], transfer learning [56], and natural language processing (NLP) [13, 43, 79, 80].

Research into how ML practitioners build and evaluate models in code has shown that ML code is highly experimental and iterative compared to conventional programming [2, 64]. This observation has generated new ways of incorporating visualization into ML development processes, e.g., enhancing computational notebooks [9, 50]. However, for all the emphasis on evaluating model behavior, there are much fewer visualization tools that evaluate a model's efficiency (e.g., latency and power consumption). The few tools that exist show model metrics, but do not inform ML practitioners of the potential efficiency improvements from the latest model optimization and compression techniques.

2.5 Visualization for Model Optimization

Compared to general model evaluation, there are few existing visualization tools for efficient ML optimization. Most work studies and surveys algorithmic techniques to compress models, such as sparsification [40]. Tooling is much less developed [42]. One of the few related visualization works to ours is CNNPruner [53], which focuses on one specific compression technique, pruning, for convolutional neural network architectures. Other work shows only static visualizations of results and features during model optimization; for example, Dotter and Ward [25] analyzed model metrics such as inference time and model size along with visualizing data clusters for a classification task, and Xie et al. [94] visualized features learned by a network as guidance to better prune redundant kernels. Model graph visualizers, such as the TensorFlow Dataflow Graph visualizer [92] and open-source tools like Netron [71], allow practitioners to inspect their models, but are not designed for the task of optimization. Most existing tools are not grounded in real-world workflows and needs of ML practitioners, nor do they factor in details about a model's efficiency and hardware metrics.

3 FORMATIVE RESEARCH: MOTIVATION AND CHALLENGES

From literature it is clear that tooling for creating efficient ML models is underdeveloped. This is in part due to the specialized nature of on-device ML: building optimized models brings all the challenges of conventional ML development, but additionally requires niche expertise in hardware knowledge and access [42].

Motivated by these challenges, we sought to explore opportunities where visualization could help. To build the right tools for model optimization, we conducted formative research to better understand the challenges and needs for creating efficient models. We first conducted a small needfinding survey with ML practitioners at Apple (Section 3.1). Then through participatory design sessions, we developed low-fidelity prototypes on practitioner data to engage them with what interactive visualization could offer (Section 3.2).

Table 1: A summary of the completed responses to the needfinding survey, including their role, primary type of ML application, and years of experience in ML.

ID	Role	ML Application	Exp.
P1	ML Manager	Deployment & Optimization	10
P2	ML Engineer	Training & Optimization	9
P3	ML Engineer	Training & Optimization	8
P4	Research Scientist	Research & Optimization	9
P5	ML Engineer	Training & Optimization	5
P6	ML Engineer	Training & Optimization	4
P7	ML Engineer	Deployment & Optimization	4
P8	Research Scientist	Research & Optimization	5
P9	ML Manager	Training & Optimization	7
P10	ML Engineer	Training & Optimization	3
P11	Research Scientist	Research & Optimization	6
P12	ML Engineer	Deployment & Optimization	5

3.1 Needfinding Survey for Efficient ML

To begin, we sent out an open-ended needfinding survey to efficient ML experts within our organization to ask what features interactive tools for model optimization should support. The survey format consisted primarily of open-ended text responses and was largely unstructured to gather diverse perspectives on optimizing models. We received 12 responses, summarized in Table 1. The participant count of our survey is lower than others within our organization because we made participation criteria strict: participants were required to be experts in efficient ML, hardware optimization, and at least one area of ML (e.g., research, model training, or deployment), to ensure the data was as relevant and informed as possible. With 12 participants, they had 75 years of experience between them. We note that this survey was conducted solely within one organization, therefore practitioners may hold organization-specific beliefs and practices [74]. However, between existing field studies on efficient ML in practice [42], the number of years of experience, and the specialized expertise shared by these participants, we are confident that our findings accurately describe current challenges within their work, and efficient ML more broadly.

With regards to what features new tooling could support, many requests were domain specific to ML model and hardware analysis, such as attributing power and memory consumption to individual ML operations executed on-device. All 12 responses (P1–P12) indicated a specific metric that they regularly inspect (e.g., model size, inference speed, memory usage, memory power). Analyzing these statistics is one of the primary routine analyses efficient ML practitioners perform. Therefore, the ability to extract these statistics from an arbitrary model and quickly load them into tools for analysis will shorten the time it takes for practitioners to visualize and optimize their models. Responses made it clear that for any tool to be successful in this work, it must support this task.

However, responses indicated that only analyzing the model and hardware statistics is not enough; ML practitioners also need to know the locations of these metrics inside models (i.e., geometrically within the compiled computational graph). Practitioners do not only want to know in aggregate how much computational budget (i.e., a

threshold for model size, latency, power or an amount of any specific resource a model is allowed to consume) their models use, but they additionally want to know specific operations within the model these aggregates are heavily weighted from. Nine responses (P1–P6, P9–P11) expressed their desire for tools to help them sort, filter, and locate the biggest "offenders" (the most computationally expensive operations). Also referred to as computational bottlenecks, these are high-value hardware operations that help practitioners minimally edit models. Since it becomes harder to have an accurate model the more optimization is applied, leaving as much of the original model intact is a desirable approach. Computational bottlenecks in this case are prime candidates for potential optimization savings that practitioners want to know about.

Another group of eight responses (P2–P7, P9, P10) expressed enthusiasm for quickly testing optimization options to see the impact on hardware metrics. Quick optimization experimentation is important, as different optimizations will have different effects on the model's metrics, and it can be hard to know what the effect of optimizing a single layer will be to the entire model. Lastly, a common theme was the inherent collaborative nature of this type of work: it requires not only ML engineers, but also hardware specialists, compiler engineers, and people with hybrid expertise who can float between these roles. These practitioners have a niche, but high-demand and hybrid skillset that cannot scale with the amount of projects they work on. Tools that help them analyze models more quickly, share the results (e.g., overall latency improvements, layer-level memory analyses, or the impact of optimization before and after its applied), and perhaps educate other ML engineers about optimization techniques can help distribute their expertise.

3.2 Participatory Design and Low-Fidelity Visualization Prototyping

Given the perspectives we found from the needfinding survey, we next wanted to gather more insight into creating efficient models by letting the survey participants interact with basic prototypes. After obtaining data from one in-development model, we built low-fidelity prototypes and visualizations to provide the ML practitioners with tangible artifacts to inspect and critique. To gather the most precise and informative qualitative feedback, it was important to prototype with real data and models.

Over the course of a month, we met weekly with the 12 participants, updating our prototypes based on both their requests and our expectation on useful features. These prototypes were often specific yet disjoint solutions to problems raised in the needfinding survey. For example, one prototype was a rich data table that showed all the different metrics that could be gathered from a model compiled to run on hardware. The practitioners (P1–P12) said this was a must-have, and appreciated quickly sorting and filtering operations to find model bottlenecks and more generally see the overall distribution of compute used within the model. This first table prototype was a direct result of the needfinding survey task where practitioners all mentioned specific metrics they wanted to gather and analyze together, as oftentimes they are making trade-offs between multiple metrics (e.g., does making the model faster in one location increase its memory usage?). Later on we added results from precomputed optimizations on the model as well, which practitioners (P2-P7, P9–P11) said was helpful in having optimized model data alongside the original model.

Another prototype was a simple dashboard that implemented basic interactive visualization techniques (e.g., brushing and linking, details on demand). Practitioners (P1–P3, P8, P11) appreciated this alternative, visual view of the data from the table, but said that they constantly are inspecting specific operation values, so the table should almost always be on screen. This dashboard prototype was then positioned as complementary.

One other prototype was a simple node-link diagram of a model's hardware operations. Practitioners (P1–P9) greatly appreciated seeing the structure of a model. We then added controls to encode nodes of the graph by different metrics to highlight where in the model certain metrics were heavily weighted. This was illuminating to the practitioners, as they had not produced a visualization like this before, but have always wanted a view to find bottleneck operations geometrically in the model, not only from statistics.

By the end of the month, we had a small collection of prototypes, ranging from data tables, dashboards, computational graphs, and others, that was sufficient for demonstrating power of interactive visualization in efficient ML development. When reviewing all the prototypes with the practitioners, they again stressed inspecting their models analytically and geometrically, and that each view gives a different perspective to their work. It was agreed upon that the foundation of a future tool should support both paradigms. These prototypes helped prioritize system capabilities during our design and development of Talaria.

3.3 Design Challenges for Model Optimization

From combining the data gathered from our needfinding survey (Section 3.1) and feedback from the low-fidelity visualization prototypes (Section 3.2), the most common and pressing challenges for optimizing ML models coalesced, which we list as (**C1–C5**) below.

- **C1. Inspecting model statistics analytically and geometrically.** Efficient ML analysis requires looking at both large amounts of tabular model statistics and large network diagrams simultaneously. It is time consuming and cumbersome, yet critical, to toggle back and forth between these two views.
- **C2. Finding model bottlenecks.** Not every piece of a model needs to be, or should be, optimized. It is hard to find computational model bottlenecks and place them in context with the global architecture.
- **C3. Interactively testing multiple model optimizations.** Tools for model compression are in their infancy, and lack interactive interfaces to support general optimization analysis. It is unclear to know how much and where to apply model optimizations to hit target metrics and computational budgets.
- **C4. Collaboratively optimizing a model.** Efficient ML work requires multiple practitioners and experts to iteratively make decisions during model development. It is difficult to keep track of shared analyses from multiple contributors.
- **C5. Accurately applying model optimizations.** Translating findings from optimization analyses into practice (e.g., applying compression to a layer in a model's training code) can be time consuming and error prone.

4 VISUALIZATION SYSTEM REQUIREMENTS AND TASK ANALYSIS

From our formative research, there is clear opportunity to help practitioners create efficient ML models. Practitioners reported that existing tools were insufficient, and expressed enthusiasm that visualization could help them develop smaller, more efficient models for on-device user experiences. Given the relatively novel domain and sparsity of work that addresses this budding area of ML, we sought to design new interactive visualizations for optimizing ML models. To inform our design, we distilled five main tasks performed by practitioners that our system should support. The tasks (**T1–T5**) below are mapped to the challenges (**C1–C5**) raised in Section 3:

- **T1.** Quickly analyze low-level model and hardware statistics to understand a model's inference (in)efficiency (**C1, C2**).
- **T2.** Interactively visualize model architecture to see its topology and to find computational performance bottlenecks in the computational graph (**C1, C2**).
- **T3.** Explore varying model optimizations and quickly examine their effect on inference efficiency, including both model-wide and targeted optimizations (**C3**).
- **T4.** Allow teams to collaboratively optimize models (**C4**).
- **T5.** Make optimizations actionable by attributing low-level hardware operations to their source code locations to help practitioners know where to implement optimizations (**C5**).

5 TALARIA INTERFACE AND SYSTEM

With the tasks identified from our formative research, we present Talaria, an interactive visualization for ML model optimization. Talaria enables ML practitioners to understand how their models perform on-device and optimize them for improved inference efficiency. The system visualizes hardware statistics through a split interface showing an interactive table and model graph. Talaria is a substantive engineering effort, containing many features that address challenges practitioners face when building efficient ML. The system is model agnostic and supports arbitrary ML tasks, such as vision, NLP, and sensing. Throughout this section, we link relevant views and features to the tasks (**T1–T5**) identified from our task analysis (Section 4).

5.0.1 System Header. The Talaria header contains top-level information about a model, including key statistics that practitioners need to know and optimize, such as the targeted inference frame rate (fps), memory power (mW), and latency (ms). The header also contains the main navigation tabs for Talaria, to switch between the specific visualizations and views described below. When switching views, the system header remains fixed in the interface.

5.1 The Table View

The first main view of the interface is the Table View (Figure 1A), a rich, interactive data table that displays the low-level hardware statistics of how a model will run (**T1**). Each row of the table corresponds to one low-level hardware task, and each column encodes different metrics. One important metric is the clock time it took for a task to run (`TOTAL TIME` column), which is dual encoded in this table as both a number and an inline sparkbar [85].

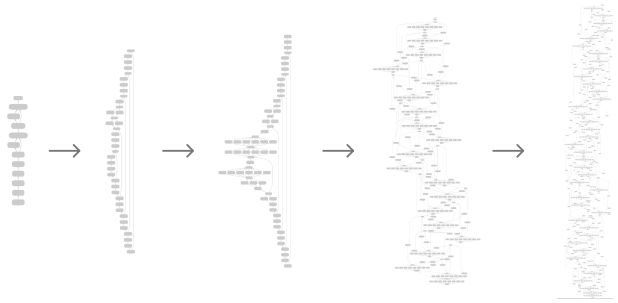

Figure 3: Five different models visualized in Talaria with increasingly complex architectures.

There are dozens of metrics to visualize, but the system displays only a few by default; the default options were chosen based on practitioners' feedback from the formative research in Section 3. Users can add, remove, or browse all the available metrics by clicking the "Visible Columns" button. Users who are not familiar with each metric can hover over the metric name in the column header to display a tooltip that describes the metric in plain language.

The Table View also supports common tasks for interacting with rich data tables that practitioners requested from our participatory design sessions. Users can sort the table by a metric when they click the arrow icon in a column header, filter the table (e.g., show tasks that took longer than 1ms), and search by the task name or ID. These features allow users to quickly explore and analyze the statistics of their models.

Lastly, the Table View is interactively linked to the Graph View. For example, selecting a task in the table will zoom in and highlight the correspond node in the graph. This is a simple but critical interaction, as it allows practitioners to link task statistics to their location in the model's graph for further analysis. Multiple selections are also supported, e.g., when the table is filtered to a subset of tasks, the Graph View highlights the selected task and auto-resizes the graph to show these tasks. This shared state is a pattern within Talaria: interactions in one view are linked with the others in the system. We decided to implement multi-coordinated views and cross-filtering from our needfinding survey since practitioners lamented that they frequently toggle back and forth between statistics and graph visualizations.

5.2 The Graph View

The second main view of the interface is the Graph View (Figure 1B), an interactive canvas that displays the compiled model architecture graph (**T2**). Each node in the graph corresponds to a low-level hardware task (e.g., a convolution or concatenation operation). It is important to note that this graph represents the operations of a model compiled onto hardware (similar to visualizing a dataflow graph [92]), not just the conventional model architecture from model definition code. The computational graph shown in Talaria is richer and often more complicated (example models growing in complexity shown in Figure 3).

Users can freely zoom and pan on the graph to inspect how their models get compiled to hardware. For details on demand, hovering over any node displays a tooltip with important metrics

that may interest practitioners during exploration. When a user wants to get more information about a particular task, selecting a node also highlights the corresponding task in the Table View, which contains all the other available metrics as discussed above. Besides selecting a single node, users can also select multiple nodes with a lasso selection; this selection also filters the Table View to the corresponding tasks in the selection.

Since models can be large, both in depth (e.g., number of layers) and width (e.g., parallel layers or branches), the Graph View shows a minimap (a small graph overview) to allow users to quickly identify areas of interest (Figure 1B). Minimap examples for five models with increasingly complex architectures are shown in Figure 3. The minimap also helps users keep the global model geometry in mind when they are zoomed into a particular region. Users can drag the minimap selection window to reposition the main Graph View (e.g., quickly jump to a farther away location in the model). The minimap can also be hidden to maximize screen space.

Another technique to wrangle large models is to group relevant tasks and construct a hierarchy when appropriate. When practitioners export models, they can define groupings in their code (e.g., group all tasks in a Transformer unit, or group tasks in a specific sub-network). With a hierarchical graph where supernodes can be interactively expanded or collapsed (taking inspiration from [92]), practitioners can reduce the number of nodes in their view to focus on higher-level model structure.

The last important feature of the Graph View is coloring the graph by a model metric. This is critical for quickly finding computational bottlenecks within a network. Users can pick a metric in either of two locations: (1) the dropdown menu in the Graph View, or (2) the "plot" icon in a column header in the Table View. Either selection updates the color of the nodes, where darker blue indicates more computationally expensive tasks, as seen in Figure 4. This design lets dark nodes (i.e., bottleneck tasks) stand out when zooming out for an overview.

Figure 4: Three examples of the Graph View encoding different hardware metrics on the same model to quickly identify potential model bottlenecks. Dark blue nodes indicate higher values for a metric, e.g., latency, memory, or power usage.

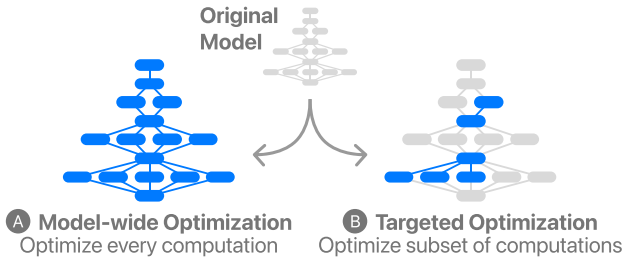

Figure 5: An illustration of two types of model optimization. (A) Model-wide optimization applies a compression technique to the entire model, regardless of outcome. (B) Targeted optimization only compresses certain model operations. Talaria supports both, and allows practitioners to interactively optimize individual model operations.

5.3 Interactive Model Optimization

In addition to visualizing model statistics and the compiled graph, Talaria contains powerful features to help ML practitioners make informed decisions on model optimization (**T3**). To optimize a model, practitioners typically have to implement and apply optimizations, such as specific compression techniques, to empirically test which techniques give the best results. This can be time consuming and feel like "searching in the dark." Instead, Talaria enables users to select and compare model optimizations in real time.

How is this possible? At compile time, Talaria precomputes many possible optimizations for every task and saves this data to the Talaria backend server. Although these are estimations of hardware metric savings (e.g., latency and power), in most of our tests, models are sufficiently accurate (within 1–3% variance, compared to actual hardware benchmarking). When a user selects an optimization, the interfaces updates in two places. First, the table in the system header shows the result on the model's overall metrics (as seen in Figure 1, where this optimization results in saving 18.02% memory power and 11.55% latency). Second, the Table View shows the new, optimized statistics for each task colored green or red depending on if they improved or regressed (Figure 1).

Talaria supports two types of optimization: (1) model-wide predefined optimizations (Figure 5A), and (2) task-specific targeted optimizations (Figure 5B).

5.3.1 Model-wide Predefined Optimizations. Model-wide optimizations are a commonly used yet blunt approach, where the same optimization technique applies to ever single task in a model. For example, one could either quantize or sparsify an entire network to reduce model size. Talaria provides predefined model-wide optimizations that are most commonly considered (Figure 6A). Since Talaria allows a user to examine optimization impact in real time, this is a great first attempt when someone wants to quickly estimate latency or power savings with common model-wide optimization.

5.3.2 Task-specific Targeted Optimizations. More advanced and novel to Talaria are targeted optimizations that apply to specific tasks, for example a bottleneck task that is computationally expensive. Whereas model-wide optimizations can be seen as coarse techniques, targeted optimizations give users fine-grain control. Targeted optimizations avoid excessive compression of a model, which better preserves behavioral metrics like accuracy.

To optimize a task, users can click the "Optimize" button in the Table View to see a modal that presents an exhaustive list of combinations of optimizing a task's "Input Format", "Output Format", "Kernel Format", and "Weight Sparsity." Each optimization also shows the impact on this task's latency and memory power. Users can filter these options to a subset of optimizations that they prefer, e.g., only considering options with `int8` kernel quantization. To help practitioners make a decision, each option's relative change among all options are colored for easier comparison. For example, in Figure 6B, green text indicates positive outcomes (e.g., latency drops) and red text indicates the opposite. While optimizing a task often leads to better inference efficiency, some optimizations make trade-offs (e.g., reducing memory but increasing latency).

With the Table View, the Graph View, and real-time optimization features, novel analysis workflows start to emerge. ML practitioners can observe metric distribution patterns in the Table View, quickly locate the model bottlenecks from the Graph View, then selectively optimize those tasks to squeeze out the best possible inference efficiency. This follows a guiding design principle where practitioners want to minimal edit and optimize their models. Talaria allows them to prioritize optimizations and get the best "bang for buck."

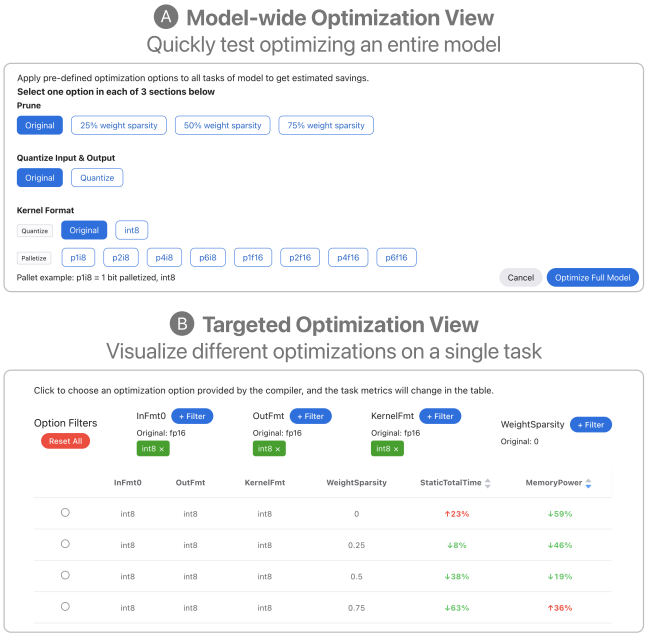

Figure 6: Talaria's (A) model-wide optimization for quick experimentation and (B) targeted optimization for compressing a single hardware operation. Targeted optimization displays a table where rows are different compression techniques, with metric changes colored green or red. In this example, a user has filtered the table to only consider optimizations where the input and output formats are quantized to `int8`.

5.4 Collaborative Optimization and Saving Compression Analyses

In practice, building ML models is a collaborative effort with multiple contributors. Talaria was designed with this workflow in mind, and contains lightweight but important features to support collaborative model optimization for ML teams (**T4**).

A user can save an optimization in Talaria by clicking the save button and providing a name for the analysis. An example can be seen in Figure 1 in the system header where a user has saved an optimization named "CHI 2024 Analysis." This feature is also useful for (1) saving an analysis as a specific checkpoint, (2) tracking the path to a particular savings goal, or (3) saving an optimization and then restarting to work on an alternative.

Moreover, when a model is uploaded to Talaria, a unique URL is generated. Once the uploader grants permission, this URL can be shared to individual users or user groups, and the model will appear in collaborators' model list page. This is designed for a common workflow, where an ML engineer optimizes their model, saves the analysis, and sends the URL to their team for review. Model owners can also enable link sharing, so that any other user could load a previously saved optimization, edit it, and save it as new analysis.

5.5 Source Code Tracking

Once an ideal optimization is chosen, practitioners need to apply it back to their code. Talaria supports a key feature called source code tracking which maps each hardware task back to the model definition in code (**T5**). To enable source code tracking, practitioners export models using Talaria's companion framework, which constructs a graph of hardware tasks. During graph construction, it parses the call stack of each API call to get code locations. The exported model package includes a JSON file mapping source code to hardware tasks. The end result is that users can trace a single task from hardware in the stack to the exact line of code of their model definition which spawned the task. Users can interact with this feature in two views: Code Locations and the Code Browser.

5.5.1 Code Locations View. Selecting a task from the Table View or the Graph View populates the Code Locations view, which shows the code snippet that spawned the task. This allows a practitioner to quickly find which code to edit to apply the optimizations.

5.5.2 Code Browser View. Each code snippet also contains the name of the file that the snippet belongs to. Clicking on the filename changes the view to the Code Browser (a read-only, web-based code editor), which highlights the line of code from the snippet to give the practitioner better code context. The code browser has common features of a code viewer, including a filetree browser, syntax highlighting, and a code minimap.

5.6 Complementary Visualizations

Talaria also contains three complementary visualizations to help practitioners explore model statistics. The visualizations show model

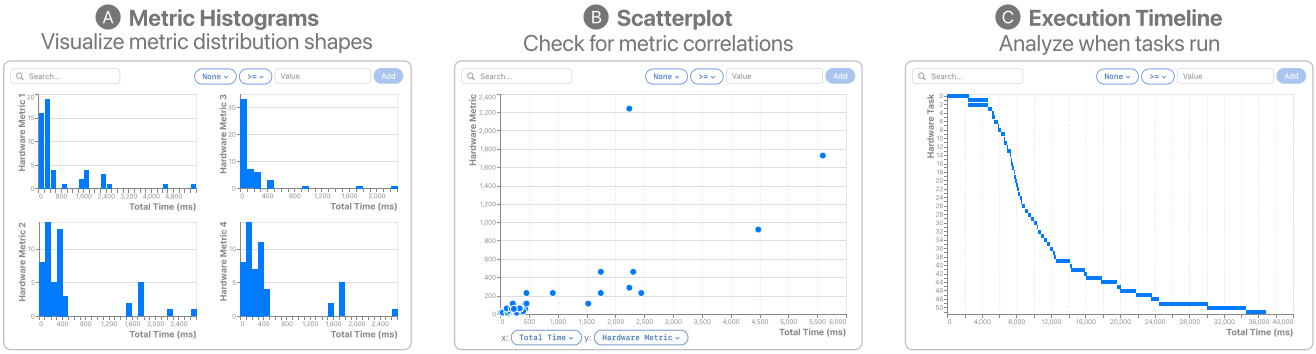

Figure 7: Complementary visualizations to help ML practitioners analyze their models. (A) The Univariate Metric Histograms give users a quick glance of the distribution shape of various model metrics. (B) The Scatterplot helps identify correlations between model metrics. (C) The Execution Timeline shows when the different operations of a model execute.

operations, i.e., rows in the Table View and nodes in the Graph View. These views are interactive and share state within the tool, e.g., selecting or filtering tasks in one view updates all other views. Users toggle between these views from tabs in the system header.

5.6.1 Metric Histograms. The first complementary view is a grid of univariate histograms (Figure 7A) to give users a quick glance at the distribution shape for every metric of their model. Lightweight interactions are available, such as a range selection to filter out parts of a distribution that are not needed; Talaria then updates the selection state of the system and remaps the axes to fit the data subset. Filtering multiple histograms helps users find a subset of tasks that they are interested in.

5.6.2 Scatterplot. The second complementary view is a scatterplot (Figure 7B) that helps users find correlations between metrics. Each axis contains a dropdown to specify a metric. Hovering over a point displays a tooltip with task details. Clicking or selecting points also selects those tasks in the other views of Talaria.

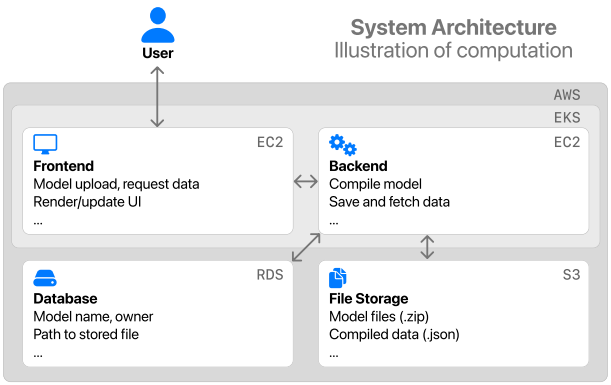

Figure 8: The Talaria system architecture. A user interacts with the web frontend to visualize the model. The frontend communicates with a backend server that compiles the model, and also connects to database and file storage services for saving and retrieving model information.

5.6.3 Execution Timeline. The third complementary view is a timeline visualization (Figure 7C) that helps users see the execution of their model's tasks chronologically. Tasks are arranged on the y-axis, and time on the x-axis, where bars indicate how long a task took. This encoding makes it easy to compare computationally expensive tasks (larger bars) to smaller tasks. Moreover, this view is useful in both quickly finding top offenders, i.e., computationally expensive tasks, and chronologically locating each task when it runs during inference time. Similar to other views, clicking any task updates the Talaria selection in the other views.

5.7 System Implementation

Talaria is a web-based system built on a common web stack. The guiding design philosophy of the system is to keep as much as the workload as possible in the browser and use a backend primarily for data and model compilation.

For the frontend, we used open-source libraries including Vue.js[1] for the primary UI framework, D3.js[2] for data transformations and visualization rendering, and the Monaco Editor[3] for displaying code. For the backend, we used Flask[4] as a lightweight WSGI app framework that communicates with our database and storage and serves data to the frontend. Most of the interactivity logic is located in the frontend (e.g., rendering and visualization interactivity), while the backend is mainly used to provide precomputed JSON data (e.g., computing possible optimizations as mentioned in 5.3). Our service is hosted on Amazon Web Services Enterprise (e.g., EC2, EKS, RDS, S3)[5]. For more details on how each component relates to one another, see our system architecture diagram in Figure 8.

6 ILLUSTRATIVE USAGE SCENARIO

To show how Talaria's features described in Section 5 work together to help ML practitioners visualize and optimize their models, we present an illustrative usage scenario.

[1] https://vuejs.org/
[2] https://d3js.org/
[3] https://microsoft.github.io/monaco-editor/
[4] https://flask.palletsprojects.com/
[5] https://aws.amazon.com/

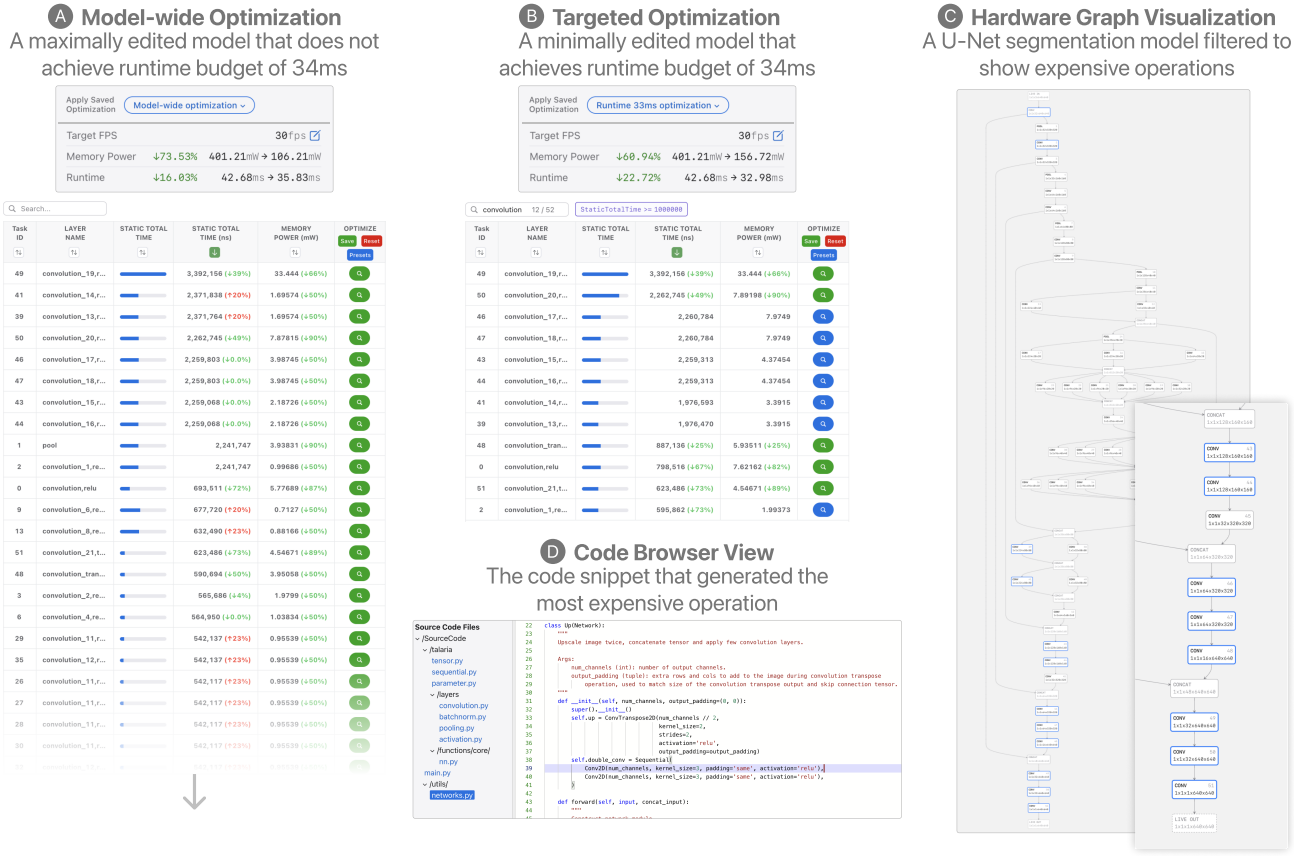

Figure 9: An illustrative usage scenario where an ML practitioner Moira must achieve a runtime budget of 34ms on a U-Net segmentation model. With Talaria, she (A) quickly tests a model-wide optimization baseline (using the quantization compression technique, but does not meet budget. Instead, she (B) filters the hardware operations to find bottleneck nodes, applies targeted quantization optimization, which meets the budget. (C) The Graph View highlights the most computationally expensive operations from the earlier filter, and the (D) Code Browser view shows which code snippet generated them.

Scenario setup: How to speed up inference of an image segmentation model? Moira is an ML engineer on a product team developing a model that will power a new feature on a mobile device. The task is image segmentation, and the team decides to use a lightweight U-net architecture [72]. Moira has been iterating on this model to get the best accuracy possible. To ship this model on-device, its inference runtime must be within budget to ensure a good user experience. To start, Moira loads the model into Talaria to benchmark its current runtime. In the system header, she reads off the top-level metrics for the model: "Memory Power: 401.21mW" and "Runtime: 42.68ms." The allowed runtime budget for this model is 34ms, so she needs to reduce the runtime by about 20%.

Visualizing model architecture on hardware. Moira first familiarizes herself with Talaria, including the two main views: the Table View and Graph View. She sees 51 rows in the Table View, corresponding to 51 model operations running on the hardware. She first wants to get a sense of how these operations are organized, so in Graph View she zooms and pans around the model to inspect the structure generated by the hardware compiler. She sees the U-Net architecture running on hardware represents her expectations: the input and output share the same size, and the two "sides of the U" (called the contracting and expansive paths [72]) are seen from the graph connections running from subsequent convolutional layers from the beginning operations to the final operations.

Quick test: Applying model-wide optimizations. When analyzing a new model, a common baseline is to try model-wide optimization: optimizing every model operation with the same compression technique. Moira wants to see if this quick test satisfies her runtime budget. She clicks the model-wide optimize button and sees multiple compression options supported by Talaria, including quantization, pruning, and palettization. Moira is mainly interested in quantization, so she chooses to cast all input, output, and kernel formats from `fp16` to `int8`. The resulting model (Figure 9A) reports top-level metrics of reducing memory power by 73.53% (401.21mW → 106.21mW) and runtime by 16.03% (42.68ms → 35.83ms). Note that there is no guarantee that optimizations always make performance

better, e.g., the overhead of optimization could be larger than the savings. In this example, the runtime of some operations (colored red in the Table View of Figure 9A) are increased. Although this is a big performance improvement, it does not achieve the runtime budget of 34ms. Before trying another optimization, Moira clicks the "Save" button and provides a name "Model-wide optimization," to keep a checkpoint of her work.

Analyzing model statistics and finding bottleneck operations. Before trying a targeted optimization, Moira needs a deeper understanding of the model performance. To inspect model statistics, she reads the Table View to examine existing operations and their runtime distribution. Scrolling through the tasks and reading down the "Layer Name" column, she sees the model is mainly composed of convolution and pooling operations. From model-wide optimization, she finds quantizing pooling layers does not reduce runtime, so she enters "convolution" in search box to focus on these operations. Since the Graph View and Table View are interactively synced, now the Graph View highlights the convolution operations with a blue border. She then sorts the convolution operations by their runtime to reveal the runtime distribution across the model. From the Table View's "Static Total Time" column, she finds twelve operations take up a majority of the total runtime. She then applies a filter to remove the operations that are less than 1ms. Once again, the Graph View updates to highlight the convolution nodes that satisfy the filter (Figure 9C). These bottleneck operations form the candidate set that Moira wishes to optimize.

Combining geometric and analytic model knowledge. Using the "Color by Hardware Stats" feature, Moira visualizes model architecture and runtime together in Graph View. This feature colors each node a shade of blue (darker means longer runtime). She confirms that the darker nodes are the operations she has filtered in the Table View, and makes the observation that they appear at the beginning and end of the model. This is a fast and powerful way to confirm and visually find model bottlenecks.

Applying targeted model optimizations. Moira now has her candidate set of operations for a targeted optimization. She clicks the optimize button for the most computationally expensive operation and sees a list of combinations of compression techniques. Moira starts with quantizing this operation by filtering the table with `int8` for the input, output, and kernel; the result shows 39% reduction of the runtime and 66% reduction of the memory power for this single operation. After selecting this option, Talaria applies the optimization and shows Moira the improvements in the table row. The top-level metrics in the system header are also updated to show that the overall memory power is reduced by 43.14% (401.21mW → 228.12mW) and the runtime is reduced by 17.45% (42.68ms → 35.23ms)—this is close but still not under the required budget (34ms). Moira tries to optimize the next most computationally expensive operation with the same quantization. Talaria updates the metrics and shows an improved memory power reduction of 60.94% (401.21mW → 156.72mW) and runtime reduction of 22.72% (42.68ms → 32.98ms). While this optimization's memory power reduction is not as strong as the model-wide optimization, her targeted optimization (Figure 9B) successfully meets her runtime budget. Note that if an operation is dependent upon other operations, Talaria handles these dependencies and optimizes the corresponding operations.[6] Before moving on, Moira clicks the "Save" button and names the analysis "Runtime 33ms optimization."

Sharing optimized models with others and evaluating on hardware. With her targeted optimization and model-wide baseline analyses completed, Moira wants to share them with her team. In Talaria, she clicks the share button to add emails of team members, who will see this model in their model lists. Moira also copies and pastes the Talaria URL into her team's chat, so others can directly access the model. Now, other team members can inspect the analysis checkpoints Moira made, fork and create their own optimizations, and share back with her. While her team inspects the results, Moira prepares her code to make the necessary modifications to apply the optimizations. To locate the code to modify, she clicks on each optimized operation, and then clicks the Code Tracking tab, which highlights the code snippet from the Python source code that generated this hardware operation. For better context, Moira clicks on the filename of the snippet to see its location in the codebase (Figure 9D). With her code updated, she now can run and evaluate the optimized model on hardware: she finds the actual runtime was reduced to 33.35%, only around a 1% difference from the predictions made by Talaria. Talaria allowed Moira to understand and experiment, in real-time, with optimizations for her segmentation model, instead of blindly applying compression techniques and waiting longer for hardware benchmarking.

7 EVALUATION: LOG ANALYTICS, USABILITY SURVEY, AND QUALITATIVE INTERVIEW

We deployed Talaria within our organization and over time gained users as multiple teams found it valuable to their work. We described the system as a new, interactive approach to help ML practitioners evaluate and optimize their model inference efficiency. Here, we report on three different evaluations (**E1–E3**):

- **E1.** A **log analysis** (Section 7.1) to track the growth of users and models in Talaria over time.
- **E2.** A **usability survey** (Section 7.2) to determine the most and least useful features to users.
- **E3.** A **qualitative interview** (Section 7.3) with the most active users to learn about their experience using the system for over time and their suggested improvements to help them create efficient ML models.

Timeline. The implementation of Talaria started in the Summer of 2021, with the first version completed in the Fall of 2021. We have been actively developing the tool since then, including adding features, providing maintenance, and talking with practitioners over 2 years. The log analysis data was captured from the Fall of 2021 to the Fall of 2023. The usability survey was sent in the Spring of 2023. Similarly, for the qualitative interview, we spoke with the power users of Talaria in the Spring of 2023.

Protocol. Our study includes three evaluations, all of which had their protocols approved by an internal IRB. Recruitment strategies

[6] For example, in Figure 9 the 0th and 49th operations are connected by a path, therefore quantizing the 49th operation's input to `int8` will update the 0th operation's output to be `int8`. Similarly, the 51st operation's input must match to `int8` due to the 50th operation's quantization.

for each evaluation are described separately in their own section. No compensation was given, as all participants were salaried employees of our organization. However, many participants were interested in learning about our results. At the end of the study, we briefed participants and their teams on our results.

7.1 Log Analytics

In this first evaluation, we analyze the backend logs of Talaria as one angle to inspect its usage and broader adoption over time. Inspecting user logs in aggregate gives us insight into the tool's adoption, performance, and user behavior patterns, which can lead to opportunities for future improvements. In our evaluation, we focus on inspecting cumulative quantities, such as the number of users logged and the number of models submitted. A deeper analysis, such as which interactions each user takes on specific UI elements, is out of scope for this work. To protect user privacy, all names have been scrubbed from the data.

After filtering out the developers of the system and models used for testing, we count 800 unique users, 161 of which have submitted at least one model (20%). This means one-fifth of users submit a model, whereas others view a model shared to them by a collaborator. Observing the cumulative number of users over time is shown in Figure 10A. Similarly, we can inspect the cumulative number of models that have been submitted. Over the same time frame, there have been 3,600+ models submitted, as shown in Figure 10B.

In both charts in Figure 10, we see an interesting pattern: there are multiple large upticks in usage at a single time. In the users chart in Figure 10A, this suggests that an entire team discovered Talaria by viewing a model that was shared with them, or a teammate was demonstrating the tool and had colleagues simultaneously log in to try it organically. Note that the largest, most recent spike happened when some models were demoed and shared to wider audiences for educational purposes. In the model chart in Figure 10B, upticks suggest that a developer submitted multiple models at once, perhaps testing different hyperparameters or architectures. These usage patterns are useful vectors for understanding how ML practitioners use Talaria, and are discussion points we follow up on below.

7.2 User Survey on Feature Usability

In our second evaluation, to understand the usability of Talaria, we surveyed users to rate the usefulness of different system features. The survey first asked for basic information about a participant's job

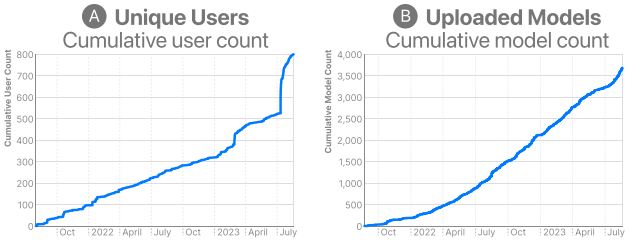

Figure 10: The cumulative number of (A) unique Talaria users (800 total) and (B) unique models submitted to Talaria over time (3,600+ submitted).

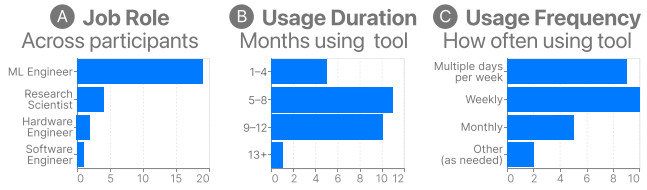

Figure 11: A summary of the usability survey participants, including their (A) job role, (B) how long they have used Talaria, and (C) how often they use Talaria.

title, role, and duration / frequency using the system. The remaining questions asked participants to rate 20 different Talaria features, grouped into the categories described in Section 5. We piloted the survey with three practitioners to ensure it took less than 5 minutes to complete. For recruitment, we sent the survey to email and chat groups specifically related to the tool's development and user base. In total we received 26 responses.

Our participants, summarized in Figure 11, include multiple types of ML practitioners (Figure 11A), including research scientists, ML engineers, and hardware engineers. They also span a wide breadth of application domains, such as ML prototyping, model training, model evaluation, hardware, and compiler design. When asked how long they have used Talaria (Figure 11B), responses ranged from 1 to 18 months. During that time, when asked how often they use Talaria (Figure 11C), responses showed most practitioners use Talaria multiple times a week or weekly, which is strong evidence that the system has been impactful to their work.

Inspecting the responses to the study in Figure 12 reveals a number of patterns. First, in general it is encouraging to see a majority of responses are positive across all feature categories. Standout features that are the most useful to practitioners include the Table View, Graph View, and interactive optimization options. While the reception to various features within the Table View are high, of the two main views it is surprising how strong the positive response is for the Graph View. This shows the power of visualization: while many optimization tasks can be solved with the Table View (e.g., sorting tasks by a particular metric to find the most computationally expensive tasks), viewing a model statistics geometrically by encoding them in the graph provides invaluable context. It is also encouraging that the complementary visualizations are rated highly useful, despite their conventional design and utility.

If we consider the features that were least useful or not applicable to users, the collaboration and and source code mapping categories stand out. While both of them have half or more of their responses being very useful, these two categories are the least used or known. We suspect that not all Talaria users are collaborating within a larger team, and some may use the tool individually. It also could be the case that a user accomplishes everything they needed within Talaria, and does not need to export any other materials. The source code mapping features having more not applicable responses is also insightful. One hypothesis here is that of the two types of optimizations, applying model-wide optimization does not require specific code edits, since the optimization simply applies to every operation; therefore a user does not need this feature. Another hypothesis is that the discoverability of these features could be

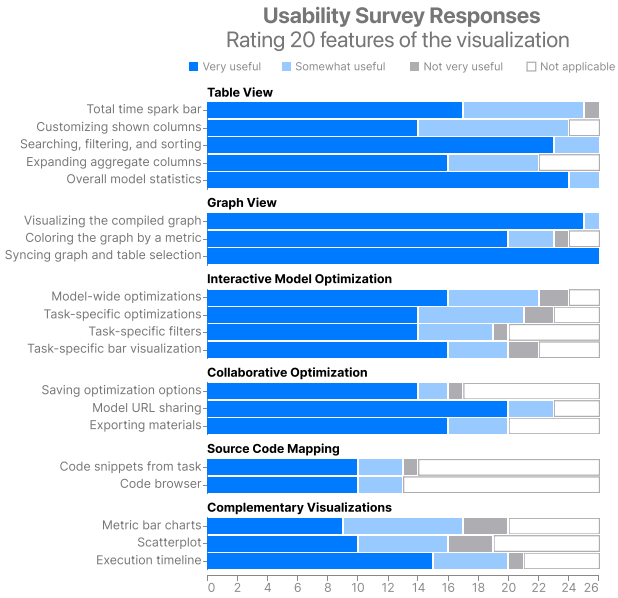

Figure 12: The responses to the usability survey grouped by feature. Participants rated 20 different features of the system.

improved, since the results show these features are useful or not applicable, only 1 of 26 response says they are not useful.

7.3 Qualitative Feedback from Power Users

In our final evaluation, we gathered feedback during several 30-minute semi-structured interviews [12, 52] with Talaria's most active users, i.e., power users, to understand their experience of visualizing and optimizing their own models. We chose a semi-structured format to ensure participants spoke to each question we prepared, with the flexibility to freely speak to their specific work and express any alternative viewpoints or opinions they may hold [52]. This method is well-suited to gather firsthand and personal knowledge of efficient ML work that was not captured or anticipated in our previous evaluations [12].

Talaria power users were found by computing the total number of models submitted by each unique user and sorting to find the ones who have submitted the most models. We interviewed 7 users, including research scientists, ML engineers, and hardware engineers. A summary of the participants can be found in Table 2. These users have interacted with Talaria the most and are already proficient using its features. We asked specific questions about their user experience, including questions to make them reflect on their own work. We also asked open-ended questions to learn about future improvements that could help them better optimize their models. For all interviews, one author led the questioning, while another took notes. With participant's approval, we recorded conversations to refer back to during analysis.

The interview questions were structured around the challenges that practitioners face with efficient ML (Section 3) and tasks we identified that tooling should support (Section 4). From the interview data, we conducted a thematic analysis method to group common workflows, user behavior, and best practices of model optimization into categories [29]. Each participant's data and transcripts were independently reviewed and manually coded using inductive coding [84].

Table 2: A summary of the participants interviewed for the qualitative interview evaluation, including their roles, primary types of ML application, and years of experience.

ID	Role	ML Application	Exp.
P1	Research Scientist	Research & Optimization	6 yrs
P2	Hardware Engineer	Deployment & Optimization	5
P3	ML Engineer	Training & Optimization	6
P4	ML Engineer	Training & Optimization	6
P5	Research Scientist	Research & Optimization	4
P6	ML Engineer	Optimization	7
P7	ML Engineer	Training & Optimization	7

7.3.1 Analytically and Visually Optimizing Models. It was exciting to learn that practitioners had their own preferences for the views they used in their analyses. Between the two main views (Table View and Graph View), their preference was nearly split: after uploading a new model, P2, P4, and P6 looked at the Table View first, whereas P1, P3, P5, and P7 considered the Graph View first. Despite this first reaction, nearly all participants mentioned that they relied on two views together for analysis (**T1**). P4 stated it plainly: *"Both the numbers and graph are equally important."* Participants told us that selecting a task in the Table View and simultaneously highlighting it in the Graph View (and vice versa) was transformative to their work. Of all the features in Talaria, P2 said this interactive selection between the views was their favorite.

One unexpected task supported by the Graph View was that practitioners used the graph to verify architecture questions they had when building a model. This is likely a potential reason that the Graph View was rated so highly in the usability survey (Section 7.2). For example, P3 said that they use the graph to confirm their understanding of an architecture change, and are then eager to see how it compiles to hardware. P2 said they view the graph as a *"quick check."* This model verification task is interesting, as it emphasizes the unique consideration of hardware details that conventional ML does not usually need to work with. To measure on-device metrics such as power, latency, and memory usage, practitioners need to know how their models will decompose into individual operations on hardware. Visualization greatly helps in this task by allowing practitioners to visually inspect the topology of their model graphs and to encode different metrics on top of the graph.

> *"I use Talaria to sketch out the topology of a model; it is a nice tool to visualize a model as well as looking at the power and perf."* — P7

7.3.2 Discovering Computational Bottlenecks. We next asked about Talaria's ability to find computational bottlenecks (**T2**), or what P2 referred to as *"top offenders"* and P7 referred to as *"hot spots"* (i.e., tasks that have the most latency, memory, or power consumption). A major goal of the Talaria design was to allow practitioners to find model bottlenecks quickly, either from low-level statistics, the model graph, or other visualizations. It was unsurprising then that

all participants said this was one of their primary reasons to use the tool, and that Talaria did it well. We dig into the bottleneck finding process by asking if practitioners had ever uploaded a model and been surprised by a bottleneck. P1 said this *"happens often,"* and P2 said this *"happens all the time."* More specifically, P3, P4, and P5 said that they have all uploaded models and found additional hardware tasks that were not supposed to be there. For example, when applying a targeted quantization to a subset of hardware tasks, practitioners found redundant data type conversions between the input and output of various hardware tasks. With Talaria, they could find these bottlenecks and fix them faster than before.

> *"The nice thing about Talaria is that it tells you stuff that you might not be expecting, but it also gives you a way to see why that was happening."* — P2

7.3.3 Faster Optimization Experimentation. Beyond visualizing model statistics and finding computational bottlenecks, we investigated how the power users engaged with the interactive optimization features (**T3**). Use cases here varied by practitioner needs. For example, P6 heavily uses the model-wide optimization. P6 works with and consults for multiple model development teams, so whenever they receive a new model, they need the fastest way to test the maximal savings to quickly share back to the teams, which can be achieved by optimizing an entire model with a particular compression technique. The other six participants more often use the targeted optimization features. Based on their applications, participants preferred different compression techniques (e.g., quantizing inputs and outputs only, quantizing kernels, or pruning weights). P3 said they appreciate that Talaria *"clearly shows me what options I have for each layer."*

> *"Talaria is nice because I can try a couple of optimization options quickly, and it can tell me at a finer level what's going on."* — P7

One unique workflow worth highlighting was from P4, where they said they prefer to do targeted optimization because they do not want to change every layer, which is more likely to cause accuracy loss. P4 instead works backwards, by applying model-wide optimization first and then removes optimizations to the sensitive layers that need to be preserved. We noted this approach to inform future users that they can optimize the full model but also selectively remove tasks that need full precision.

7.3.4 Optimizing Models within Teams. We also asked about the practitioners experience using Talaria in a collaborative setting (**T4**). From the interviews, it was clear that sharing is heavily used, but we also wanted to better understand the model receivers: are they modeling engineers, hardware experts, or broader stakeholders? When sharing Talaria URLs within their own team, P2 said they will iterate on models individually and then share the best model as final proof of their work. P7 has a similar workflow, where when they receive a new model, they upload it to Talaria, then send back a Talaria URL to their collaborators, saying: *"This is what you originally had, and here's what I got it down too."* P3 and P5 said they will share multiple URLs (different versions of a model) to their teams for comparison. P4 and P6 said that compared to only reporting top-level metrics, it can be more valuable to share Talaria URLs in case a stakeholder wants to go deeper.

Lastly, P1 recounted a scenario where they were consulting for reducing model latency. They found themselves in-between a modeling team and a hardware team, and regularly shared Talaria URLs to both teams to explain changes and potential savings. P1, an efficient ML expert, explained that they regularly consult on projects that need to hit tight budgets to produce the best user experience. While they gladly share their expertise, this approach is not scalable, especially as the number of projects grow. They were excited to see interactive tools, such as Talaria, help others without this expertise optimize their own models.

> *"Since some people have [efficient ML] tribal knowledge, [...] self-service is definitely the future."* — P6

7.3.5 Closing the Loop: Applying Optimizations. Lastly, we report on practitioners taking their optimization analysis and applying it back to their codebase (**T5**). Recall in the usability survey this feature category was the least used (Figure 12). This result is also reflected in our interviews, where practitioners did not have as many examples to describe. Our original intent was that practitioners have an actionable next step after using Talaria. Our novel contribution here is attributing individual hardware operations back to source code. However, practitioners explained that applying optimizations to code is only one iteration they might do. Other iterations a practitioner might do may be trying a different architecture, updating the model compiler, or exporting statistics to run their own additional analysis outside of Talaria. We believe there is opportunity here to further improve the ML developer experience, however, what is most important is that our users did not get stuck when using Talaria, and that the system gave them something actionable to do next, even if it was not within the system itself.

> *"Ultimately Talaria helps in creating models that run faster, while being more friendly to the developer."* — P6

8 DISCUSSION: LIMITATIONS AND FUTURE WORK FOR OPTIMIZATION VISUALIZATION

8.1 Model Comparison

From our log analysis in Section 7.1, we observed a particular user behavior: ML practitioners may submit multiple versions of a model at once for comparison. A limitation of Talaria is that it only visualizes one model at a time; however, ML development is highly iterative and experimental [2, 64], requiring practitioners to compare model statistics, architectures, and hyperparameters. Efficient ML work adds another piece to this puzzle, as practitioners also need to consider trade-offs between hardware metrics, such as model size, power, and latency. From our qualitative study in Section 7.3, users want to compare models across multiple facets. Example comparisons include comparing an optimized model to a non-optimized model, comparing different compression strategies, or comparing models with different architectures altogether. This introduces new challenges: how should models be compared, e.g., against a common baseline or against one another? How do we effectively visualize relevant differences between models? What if a user wants to compare more than two models?

Since this observed workflow was so important and prevalent, after our study analysis concluded we implemented a new prototype view into Talaria called the model Diff View. While this view does

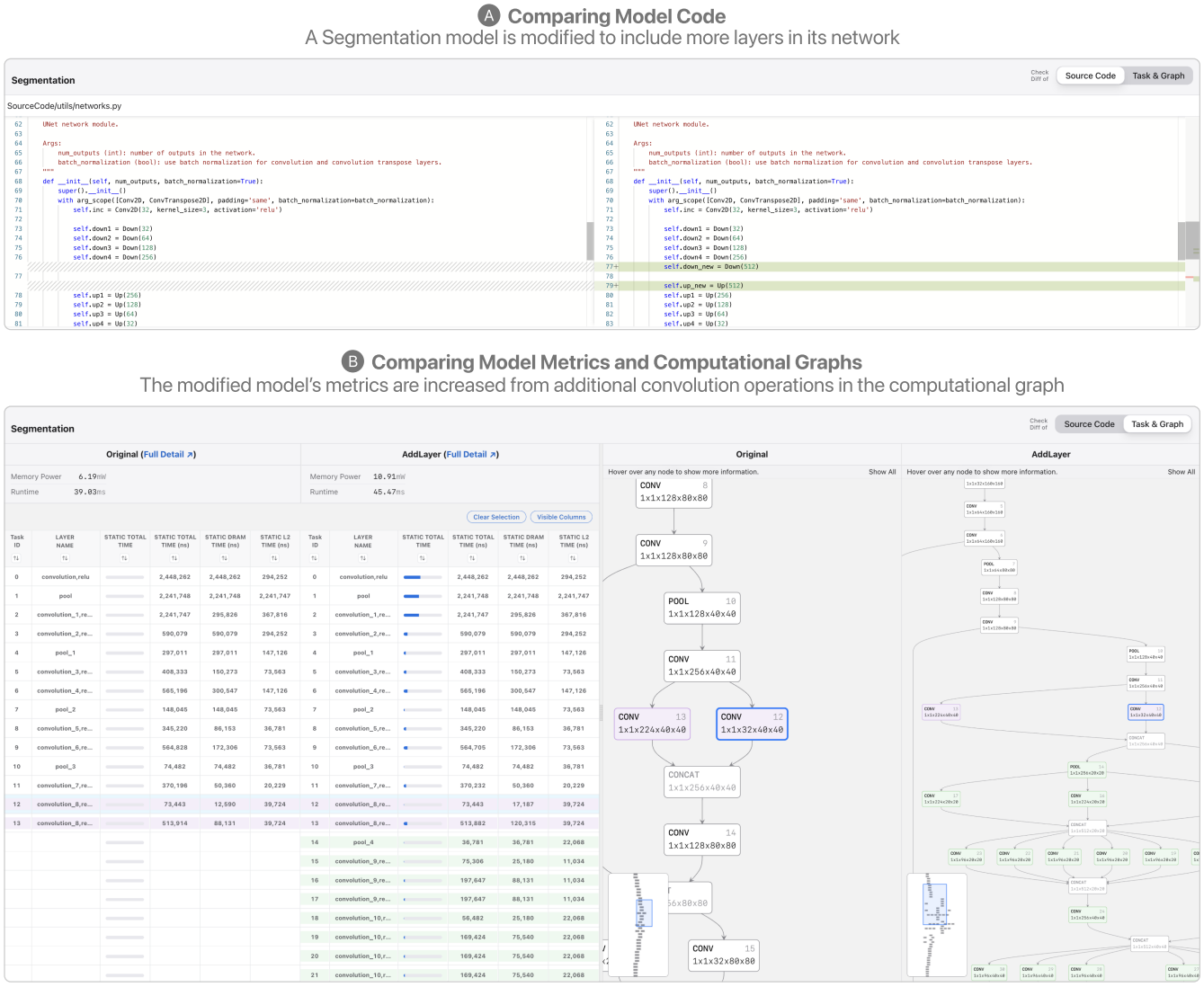

Figure 13: The prototype model Diff View added after observing practitioners from our evaluation comparing multiple models in Talaria. In this example, (A) a "Segmentation" model's code is modified to include additional layers in its network. (B) The new view shows both the original model and the modified model's hardware statistics and computational graphs, highlighting new operations in green. This new model adds multiple convolutional layers to the graph, which increases the memory power from 6.19mW to 10.91mW, and the runtime from 39.03ms to 45.47ms.

not fully support arbitrary and flexible model comparison, it does help practitioners with the common task of comparing two models, their hardware metrics, and their computational graphs, against one another. As seen in Figure 13A, the code for a model on the left is modified, and a new model with additional layers is created on the right. With both models loaded into Talaria, the Diff View now divides the main interface into four sections: two tables on left and two computational graphs on the right, mimicking the Table View and Graph View for inspecting a single model. In the updated Table View, Figure 13B shows new layers that are not present in the original model highlighted in green, and layers that were removed highlighted in red (none present in this example). Similarly, the updated Graph View shows both computational graphs, with new hardware operations colored green. With this new view, practitioners can see what impacts different model architectures have on their top-level metrics, and where modified hardware operations are located in the model's computational graph.

This is an early exploration into model comparison for ML optimization. It is important to note that model comparison visualization is not a new topic and has been explored in other tools [22, 48, 95]. However, given the size and complexity of modern ML models, improved visualizations for model comparison is worth revisiting, especially for the new challenges and constraints brought with efficient ML.

8.2 Automatic Code Editing and Interactive Model Playgrounds

Talaria allows users to test various optimization options and inspect their impact on inference efficiency. However, right now a practitioner must still manually apply those optimizations in their code. Talaria, or future tools for model compression, could automatically apply the specified optimizations in code (possibly using large language models pretrained for coding tasks [30, 62, 63]), recompile them to the targeted hardware, and visualize the results. Drawing inspiration from fluid end-user programming tools that sync code and GUI states [50], we propose an interactive playground where users upload their initial model definition code, iteratively apply optimizations, recompile their models, and finally use the optimized model code for retraining.

Lastly, given that Talaria contains both a model's code and available optimization options, there is opportunity to automatically suggest recommended compression techniques to try first. Recommending compression techniques may sound appropriate for an automated optimization algorithm. However, fully automating model optimization is not yet possible, due to how many considerations must be made both about the model and the design of the user experience the model will enable [42]. Nevertheless, future tools could enable mixed-initiative interaction and guided experimentation, where Talaria could have the power to recommend optimization options in the interface to a user and make changes to a model's source code. These feature additions could save practitioners a significant amount of time, providing more opportunities to iterate on their models.

8.3 Including Model Behavioral Metrics

Talaria's focuses on improving the inference efficiency of ML models running on-device. While it is possible to apply maximal compression to extremely optimize model efficiency and hardware metrics (e.g., model size, latency, and power), it may negatively impact the model's behavioral metrics (e.g., accuracy, precision, recall). The holistic goal of building efficient models is to find a balance between inference efficiency and an acceptable accuracy regression. One limitation of Talaria is that it currently does not take into account model behavioral metrics such as accuracy, and instead focuses specifically on the new and novel challenges brought with efficient ML work. Today with Talaria, a practitioner could quickly apply maximal optimization and minimal optimization to a model, then retrain them with these optimization configurations to check how the accuracy or other behavioral metrics changed. However, there is great opportunity to combine Talaria more deeply with model evaluation tools that visualize behavioral metrics across different subgroups of data (e.g., to catch potential fairness or accessibility concerns).

Certain technical challenges will need to be addressed to do these evaluations in real-time for interactivity, since considering behavioral metrics requires a forward pass of one's testing data through the model to compute predictions. Depending on the size of the test set, or the size of the model, this may take on the order of minutes to hours. Perhaps applying bootstrap sampling methods to create "efficient ML test sets" that a model could predict over in seconds would allow future tools to test certain model optimizations and get both behavioral and hardware metrics in real-time. This potential combination would allow ML practitioners to easily see the impact that compression methods have on behavioral metrics and inference efficiency simultaneously.

8.4 Collaborative Model Optimization

While Talaria enables practitioners to save optimization experiments and share them with others, its collaborative features are lightweight compared to other feature sets. Section 7.2 shows that the existing features are rated highly useful, but this is only a first step in the direction of collaborative, efficient ML. Collaboration in data science is not a new topic. Popular programming tools have embraced collaborative features, such as Juypter [51], Google Colab [11], and VSCode [59], and previous work has profiled how data scientist work collaboratively, both in interpersonal relationships and with tools [69, 97]. Talaria supports collaborative tooling design highlighted by Zhang et al. [97] by capturing the end result of an analysis with code and documentation (e.g., saving shareable optimization analyses and model metadata), but future extensions could see additional support for tracking a full history of one's analysis [39, 49]. Historical, collaborative features could help others reproduce an optimization step-by-step to support better reproducibility—a critical challenge due to the iterative, empirical nature of ML work [2, 64] that model optimization further complicates with additional dimensions such as compiler versions, hardware targets, and compression techniques.

8.5 Scaling Visualization Design

Talaria was built with scalability in mind, particularly for large, modern ML models. While we have not done an exhaustive scalability test, Talaria has been used for models with thousands of tasks/graph nodes and runs smoothly. The Table View only renders rows within the browser's viewport, making scrolling, sorting, filtering, and searching in real time possible even for large models. Zooming and panning on the the Graph View is fast, since the graph is rendered on canvas using WebGL and runs at a high refresh rate (e.g., 60fps) even with thousands of nodes.

However, we have tested some models that had tens of thousands of hardware operations. In these models, the Graph View was usable, but the bigger challenge in navigating the graph was that it was too large to get an intuitive sense of how the model compiled onto hardware. A good example of this is visualizing a transformer model, where the thousands of operations could be alternatively represented as a handful of sequential transformer modules. In this regime of scale, future visualization and interaction design could help, for example, by exploiting repeatable hardware operation types and automatically grouping them into supernodes (similar to [92]). While users can define their own groups in code before submitting models to Talaria, in the future groups could be constructed automatically based on exploiting repeatable hardware operations, either in sequence such as multiple convolution operations, or mined as patterns across a model (e.g., a parallel convolution structure that concatenates into a pooling operation).

8.6 Future Tools for Efficient ML

The goal of this work was to show evidence of how interactive tooling for ML optimization can be highly productive in practice. Reflecting on our evaluations, one characteristic that stands out from Talaria compared to previous work is the effort to unify the existing scripts, views, and ad-hoc analyses of practitioner workflows into single system paid off. Talaria lowers the barrier to efficient ML work and makes optimization estimation easier (e.g., clicking a button), helping people inspect the trade-offs between multiple model optimizations. This holistic view of efficient ML work, combining hardware and software, is a key differentiator between Talaria and existing work.

The design of Talaria was guided by our formative research with expert ML practitioners. We followed known visualization design patterns [14], such as implementing multi-coordinated views, cross-filtering, and Schneiderman's mantra [77] for overview + detail and focus + context techniques [21] for mixed-initiative user interfaces [44]. Despite having rigorous strategies for designing interfaces, we emphasize that tooling in efficient ML is currently underdeveloped and underexplored [42]. The few related tools focus on explaining the inner workings of a particular compression algorithm (Section 2.5). While existing work advances our understanding of specific techniques, they may not be generalizable enough for many real-world applications. Future work on designing tools for efficient ML have abundant opportunity for building on top of rich literature in HCI and visualization to advance the state-of-the-art.

9 CONCLUSION

By focusing on creating on-device and efficient models, we can design new and intelligent ML user experiences. This direction of research, while growing, is still in its infancy. More specifically, tooling for creating and optimizing models is underdeveloped. To help ML practitioners create efficient models, we designed and developed Talaria, an interactive visualization system, alongside ML experts at Apple that specialize in developing on-device models. Our visualization system enables ML practitioners to analyze models across a variety of low-level statistics, interact with a model's computational graph, and experiment with model optimizations on hardware. We hope our work emphasizes the need and importance of tooling for model optimization, and inspires future work on interactive tooling for creating efficient ML user experiences.

ACKNOWLEDGMENTS

The authors thank our colleagues at Apple for their energy, support, and guidance over this work. We especially thank Sam Xu, Matthew Kay Fei Lee, Patrick Dong, and Hojin Kee for their technical expertise. We also thank those who took time to participant in our system evaluations.

REFERENCES

[1] Yongsu Ahn and Yu-Ru Lin. 2019. Fairsight: Visual analytics for fairness in decision making. *IEEE Transactions on Visualization and Computer Graphics* 26, 1 (2019), 1086–1095.

[2] Saleema Amershi, Andrew Begel, Christian Bird, Robert DeLine, Harald Gall, Ece Kamar, Nachiappan Nagappan, Besmira Nushi, and Thomas Zimmermann. 2019. Software engineering for machine learning: A case study. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice*. IEEE, 291–300. https://doi.org/10.1109/icse-seip.2019.00042

[3] Saleema Amershi, Max Chickering, Steven M Drucker, Bongshin Lee, Patrice Simard, and Jina Suh. 2015. Modeltracker: Redesigning performance analysis tools for machine learning. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. 337–346.

[4] Apple. 2021. On-device panoptic segmentation for camera using transformers. *Machine Learning Research* (2021). https://machinelearning.apple.com/research/panoptic-segmentation

[5] Apple. 2022. Deploying transformers on the Apple Neural Engine. *Machine Learning Research* (2022). https://machinelearning.apple.com/research/neural-engine-transformers

[6] Apple. 2022. A multi-task neural architecture for on-device scene analysis. *Machine Learning Research* (2022). https://machinelearning.apple.com/research/on-device-scene-analysis

[7] Apple. 2023. *Optimizing models - Core ML Tools overview.* https://coremltools.readme.io/docs

[8] Colby R Banbury, Vijay Janapa Reddi, Max Lam, William Fu, Amin Fazel, Jeremy Holleman, Xinyuan Huang, Robert Hurtado, David Kanter, Anton Lokhmotov, et al. 2020. Benchmarking tinyml systems: Challenges and direction. *arXiv preprint arXiv:2003.04821* (2020).

[9] Alex Bäuerle, Ángel Alexander Cabrera, Fred Hohman, Megan Maher, David Koski, Xavier Suau, Titus Barik, and Dominik Moritz. 2022. Symphony: Composing interactive interfaces for machine learning. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM. https://doi.org/10.1145/3491102.3502102

[10] Donald Bertucci, Md Montaser Hamid, Yashwanthi Anand, Anita Ruangrotsakun, Delyar Tabatabai, Melissa Perez, and Minsuk Kahng. 2022. DendroMap: Visual exploration of large-scale image datasets for machine learning with treemaps. *IEEE Transactions on Visualization and Computer Graphics* (2022).

[11] Ekaba Bisong and Ekaba Bisong. 2019. Google colaboratory. *Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners* (2019), 59–64.

[12] Carolyn Boyce and Palena Neale. 2006. *Conducting in-depth interviews: A guide for designing and conducting in-depth interviews for evaluation input.* Vol. 2. Pathfinder International Watertown, MA.

[13] Richard Brath, Daniel Keim, Johannes Knittel, Shimei Pan, Pia Sommerauer, and Hendrik Strobelt. 2023. The role of interactive visualization in explaining (large) NLP models: From data to inference. *arXiv preprint arXiv:2301.04528* (2023).

[14] Matthew Brehmer and Tamara Munzner. 2013. A multi-level typology of abstract visualization tasks. *IEEE Transactions on Visualization and Computer Graphics* 19, 12 (2013), 2376–2385.

[15] Ángel Alexander Cabrera, Will Epperson, Fred Hohman, Minsuk Kahng, Jamie Morgenstern, and Duen Horng Chau. 2019. FairVis: Visual analytics for discovering intersectional bias in machine learning. In *IEEE Conference on Visual Analytics Science and Technology*. IEEE, 46–56.

[16] Ángel Alexander Cabrera, Erica Fu, Donald Bertucci, Kenneth Holstein, Ameet Talwalkar, Jason I. Hong, and Adam Perer. 2023. Zeno: An interactive framework for behavioral evaluation of machine learning. In *CHI Conference on Human Factors in Computing Systems* (Hamburg, Germany). Association for Computing Machinery, New York, NY, USA, 22 pages. https://doi.org/10.1145/3544548.3581268

[17] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. 2018. Model compression and acceleration for deep neural networks: The principles, progress, and challenges. *IEEE Signal Processing Magazine* 35, 1 (2018), 126–136. https://doi.org/10.1109/msp.2017.2765695

[18] Minsik Cho, Keivan A. Vahid, Saurabh Adya, and Mohammad Rastegari. 2022. Differentiable k-means clustering layer for neural network compression. In *International Conference on Learning Representations*. https://arxiv.org/abs/2108.12659

[19] Jaegul Choo, Hanseung Lee, Jaeyeon Kihm, and Haesun Park. 2010. iVisClassifier: An interactive visual analytics system for classification based on supervised dimension reduction. In *2010 IEEE Symposium on Visual Analytics Science and Technology*. IEEE, 27–34.

[20] Tejalal Choudhary, Vipul Mishra, Anurag Goswami, and Jagannathan Sarangapani. 2020. A comprehensive survey on model compression and acceleration. *Artificial Intelligence Review* 53, 7 (2020), 5113–5155. https://doi.org/10.1007/s10462-020-09816-7

[21] Andy Cockburn, Amy Karlson, and Benjamin B Bederson. 2009. A review of overview+detail, zooming, and focus+context interfaces. *ACM Computing Surveys (CSUR)* 41, 1 (2009), 1–31.

[22] Subhajit Das and Alex Endert. 2020. LEGION: visually compare modeling techniques for regression. In *2020 Visualization in Data Science*. IEEE, 12–21.

[23] Lei Deng, Guoqi Li, Song Han, Luping Shi, and Yuan Xie. 2020. Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proc. IEEE* 108, 4 (2020), 485–532. https://doi.org/10.1109/jproc.2020.2976475

[24] Sauptik Dhar, Junyao Guo, Jiayi Liu, Samarth Tripathi, Unmesh Kurup, and Mohak Shah. 2021. A survey of on-device machine learning: An algorithms and learning theory perspective. *ACM Transactions on Internet of Things* 2, 3 (2021), 1–49. https://doi.org/10.1145/3450494

[25] Marissa Dotter and Chris M Ward. 2018. Visualizing compression of deep learning models for classification. In *2018 IEEE Applied Imagery Pattern Recognition Workshop (AIPR)*. IEEE, 1–8.

[26] Farah Fahim, Benjamin Hawks, Christian Herwig, James Hirschauer, Sergo Jindariani, Nhan Tran, Luca P Carloni, Giuseppe Di Guglielmo, Philip Harris, Jeffrey Krupa, et al. 2021. hls4ml: An open-source codesign workflow to empower scientific low-power machine learning devices. (2021). arXiv:2103.05579

[27] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. 2021. A survey of quantization methods for efficient neural network inference. *arXiv* (2021). arXiv:2103.13630

[28] Charlie Giattino, Edouard Mathieu, Veronika Samborska, Julia Broden, and Max Roser. 2022. Artificial intelligence. *Our World in Data* (2022). https://ourworldindata.org/artificial-intelligence.

[29] Graham R Gibbs. 2007. Thematic coding and categorizing. *Analyzing Qualitative Data* 703 (2007), 38–56.

[30] Github. 2021. *Copilot.* https://github.com/features/copilot

[31] Google. 2019. *QKeras.* https://github.com/google/qkeras

[32] Google. Accessed 2022. Why on-device machine learning? *Google Developers* (Accessed 2022). https://developers.google.com/learn/topics/on-device-ml/learn-more

[33] Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. 2021. Knowledge distillation: A survey. *International Journal of Computer Vision* 129, 6 (2021), 1789–1819. https://doi.org/10.1007/s11263-021-01453-z

[34] Liang Gou, Lincan Zou, Nanxiang Li, Michael Hofmann, Arvind Kumar Shekar, Axel Wendt, and Liu Ren. 2020. VATLD: A visual analytics system to assess, understand and improve traffic light detection. *IEEE Transactions on Visualization and Computer Graphics* 27, 2 (2020), 261–271.

[35] Renjie Gu, Chaoyue Niu, Fan Wu, Guihai Chen, Chun Hu, Chengfei Lyu, and Zhihua Wu. 2021. From server-based to client-based machine learning: A comprehensive survey. *Comput. Surveys* 54, 1 (2021), 1–36. https://doi.org/10.1145/3424660

[36] Jochen Görtler, Fred Hohman, Dominik Moritz, Kanit Wongsuphasawat, Donghao Ren, Rahul Nair, Marc Kirchner, and Kayur Patel. 2022. Neo: Generalizing confusion matrix visualization to hierarchical and multi-output labels. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM. https://doi.org/10.1145/3491102.3501823

[37] Song Han, Huizi Mao, and William J Dally. 2016. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. (2016).

[38] Awni Hannun, Jagrit Digani, Angelos Katharopoulos, and Ronan Collobert. 2023. *MLX: Efficient and flexible machine learning on Apple silicon.* https://github.com/ml-explore

[39] Andrew Head, Fred Hohman, Titus Barik, Steven M Drucker, and Robert DeLine. 2019. Managing messes in computational notebooks. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 1–12.

[40] Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. 2021. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *Journal of Machine Learning Research* 22, 241 (2021), 1–124.

[41] Fred Hohman, Minsuk Kahng, Robert Pienta, and Duen Horng Chau. 2018. Visual analytics in deep learning: An interrogative survey for the next frontiers. *IEEE Transactions on Visualization and Computer Graphics* (2018). https://doi.org/10.1109/TVCG.2018.2843369

[42] Fred Hohman, Mary Beth Kery, Donghao Ren, and Dominik Moritz. 2024. Model compression in practice: Lessons learned from practitioners creating on-device machine learning experiences. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM. https://doi.org/10.1145/3613904.3642109

[43] Benjamin Hoover, Hendrik Strobelt, and Sebastian Gehrmann. 2019. exbert: A visual analysis tool to explore learned representations in transformers models. *arXiv preprint arXiv:1910.05276* (2019).

[44] Eric Horvitz. 1999. Principles of mixed-initiative user interfaces. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*. 159–166.

[45] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv* abs/1704.04861 (2017). arXiv:1704.04861

[46] Google Inc. 2021. *Know Your Data.* https://knowyourdata.withgoogle.com/

[47] Intel. 2020. *Neural Compressor.* https://github.com/intel/neural-compressor

[48] Minsuk Kahng, Dezhi Fang, and Duen Horng Chau. 2016. Visual exploration of machine learning results using data cube analysis. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*. 1–6.

[49] Mary Beth Kery, Bonnie E John, Patrick O'Flaherty, Amber Horvath, and Brad A Myers. 2019. Towards effective foraging by data scientists to find past analysis choices. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 1–13.

[50] Mary Beth Kery, Donghao Ren, Fred Hohman, Dominik Moritz, Kanit Wongsuphasawat, and Kayur Patel. 2020. mage: Fluid moves between code and graphical work in computational notebooks. In *Proceedings of the ACM Symposium on User Interface Software and Technology*. ACM. https://doi.org/10.1145/3379337.3415842

[51] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian E Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica B Hamrick, Jason Grout, Sylvain Corlay, et al. 2016. Jupyter Notebooks-a publishing format for reproducible computational workflows. *Elpub* 2016 (2016), 87–90.

[52] Eleanor Knott, Aliya Hamid Rao, Kate Summers, and Chana Teeger. 2022. Interviews in the social sciences. *Nature Reviews Methods Primers* 2, 1 (2022), 1–15.

[53] Guan Li, Junpeng Wang, Han-Wei Shen, Kaixin Chen, Guihua Shan, and Zhonghua Lu. 2020. Cnnpruner: Pruning convolutional neural networks with visual analytics. *IEEE Transactions on Visualization and Computer Graphics* 27, 2 (2020), 1364–1373.

[54] He Li, Kaoru Ota, and Mianxiong Dong. 2018. Learning IoT in edge: Deep learning for the nternet of Things with edge computing. *IEEE network* 32, 1 (2018), 96–101.

[55] Wei Yang Bryan Lim, Nguyen Cong Luong, Dinh Thai Hoang, Yutao Jiao, Ying-Chang Liang, Qiang Yang, Dusit Niyato, and Chunyan Miao. 2020. Federated learning in mobile edge networks: A comprehensive survey. *IEEE Communications Surveys & Tutorials* 22, 3 (2020), 2031–2063. https://doi.org/10.1109/comst.2020.2986024

[56] Yuxin Ma, Arlen Fan, Jingrui He, Arun Reddy Nelakurthi, and Ross Maciejewski. 2020. A visual analytics framework for explaining and diagnosing transfer learning processes. *IEEE Transactions on Visualization and Computer Graphics* 27, 2 (2020), 1385–1395.

[57] Gaurav Menghani. 2023. Efficient deep learning: A survey on making deep learning models smaller, faster, and better. *Comput. Surveys* 55, 12 (2023), 1–37.

[58] Microsoft. 2021. *Neural network intelligence.* https://github.com/microsoft/nni

[59] Microsoft. 2023. *Visual studio code.* https://code.visualstudio.com/

[60] MG Sarwar Murshed, Christopher Murphy, Daqing Hou, Nazar Khan, Ganesh Ananthanarayanan, and Faraz Hussain. 2021. Machine learning at the network edge: A survey. *Comput. Surveys* 54, 8 (2021), 1–37. https://doi.org/10.1145/3469029

[61] NVIDIA. 2023. NVIDIA deep learning TensorRT documentation. https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.html#optimize-performance

[62] OpenAI. 2021. *OpenAI Codex.* https://openai.com/blog/openai-codex

[63] OpenAI. 2023. GPT-4 technical report. *arXiv* (2023). arXiv:2303.08774

[64] Kayur Patel, James Fogarty, James A Landay, and Beverly Harrison. 2008. Investigating statistical machine learning as a tool for software development. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 667–676. https://doi.org/10.1145/1357054.1357160

[65] Antonio Polino, Razvan Pascanu, and Dan Alistarh. 2018. Model compression via distillation and quantization. *arXiv* (2018). arXiv:1802.05668

[66] PyTorch. 2018. *Quantization.* https://pytorch.org/docs/stable/quantization.html

[67] PyTorch. 2019. *Sparisty.* https://pytorch.org/docs/stable/sparse.html

[68] PyTorch. 2023. *PyTorch Examples.* https://pytorch.org/tutorials/

[69] Bernadette M Randles, Irene V Pasquetto, Milena S Golshan, and Christine L Borgman. 2017. Using the Jupyter notebook as a tool for open science: An empirical study. In *2017 ACM/IEEE Joint Conference on Digital Libraries (JCDL)*. IEEE, 1–2.

[70] Donghao Ren, Saleema Amershi, Bongshin Lee, Jina Suh, and Jason D Williams. 2016. Squares: Supporting interactive performance analysis for multiclass classifiers. *IEEE Transactions on Visualization and Computer Graphics* 23, 1 (2016), 61–70.

[71] Lutz Roeder. 2017. *Netron, visualizer for neural network, deep learning, and machine learning models.* https://doi.org/10.5281/zenodo.5854962

[72] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention*. Springer, 234–241.

[73] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and pattern Recognition*. 4510–4520. https://doi.org/10.1109/cvpr.2018.00474

[74] Edgar H Schein. 1990. *Organizational culture*. Vol. 45. American Psychological Association.

[75] David Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, and Michael Young. 2014. Machine learning: The high interest credit card of technical debt. *Google* (2014).

[76] Abhishek Sehgal and Nasser Kehtarnavaz. 2019. Guidelines and benchmarks for deployment of deep learning models on smartphones as real-time apps. *Machine Learning and Knowledge Extraction* 1, 1 (2019), 450–465.

[77] Ben Shneiderman. 1996. The eyes have it: A task by data type taxonomy for information visualizations. In *Proceedings 1996 IEEE Symposium on Visual Languages*. IEEE, 336–343.

[78] Stanford. 2023. The AI index report: Measuring trends in artificial intelligence. https://aiindex.stanford.edu/report/

[79] Hendrik Strobelt, Sebastian Gehrmann, Hanspeter Pfister, and Alexander M Rush. 2017. LSTMVis: A tool for visual analysis of hidden state dynamics in recurrent neural networks. *IEEE Transactions on Visualization and Computer Graphics* 24, 1 (2017), 667–676.

[80] Hendrik Strobelt, Albert Webson, Victor Sanh, Benjamin Hoover, Johanna Beyer, Hanspeter Pfister, and Alexander M Rush. 2022. Interactive and visual prompt engineering for ad-hoc task adaptation with large language models. *IEEE Transactions on Visualization and Computer Graphics* 29, 1 (2022), 1146–1156.

[81] Mingxing Tan and Quoc Le. 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*. PMLR, 6105–6114. arXiv:1905.11946

[82] TensorFlow. 2018. *Introducing the Model Optimization Toolkit for TensorFlow.* https://blog.tensorflow.org/2018/09/introducing-model-optimization-toolkit.html

[83] TensorFlow. 2020. *Quantization aware training with TensorFlow Model Optimization Toolkit - performance with accuracy.* https://blog.tensorflow.org/2020/04/quantization-aware-training-with-tensorflow-model-optimization-toolkit.html

[84] David R Thomas. 2003. A general inductive approach for qualitative data analysis. *American Journal of Evaluation* 27, 2 (2003), 237–246.

[85] Edward R Tufte. 1986. The visual display of quantitative information. (1986).

[86] Pavan Kumar Anasosalu Vasu, James Gabriel, Jeff Zhu, Oncel Tuzel, and Anurag Ranjan. 2022. An improved one millisecond mobile backbone. *arXiv preprint arXiv:2206.04040* (2022).

[87] Pavan Kumar Anasosalu Vasu, James Gabriel, Jeff Zhu, Oncel Tuzel, and Anurag Ranjan. 2023. FastViT: A Fast Hybrid Vision Transformer using Structural Reparameterization. *arXiv preprint arXiv:2303.14189* (2023).

[88] Pablo Villalobos, Jaime Sevilla, Tamay Besiroglu, Lennart Heim, Anson Ho, and Marius Hobbhahn. 2022. Machine learning model sizes and the parameter gap. arXiv:2207.02852 [cs.LG]

[89] Pete Warden and Daniel Situnayake. 2019. *Tinyml: Machine learning with tensorflow lite on arduino and ultra-low-power microcontrollers.* O'Reilly Media.

[90] Megan Maher Welsh, David Koski, Miguel Sarabia, Niv Sivakumar, Ian Arawjo, Aparna Joshi, Moussa Doumbouya, Luca Suau, Xavierand Zappella, and Nicholas Apostoloff. 2023. *Data and Network Introspection Kit.* https://github.com/apple/dnikit

[91] James Wexler, Mahima Pushkarna, Tolga Bolukbasi, Martin Wattenberg, Fernanda Viégas, and Jimbo Wilson. 2019. The what-if tool: Interactive probing of machine learning models. *IEEE transactions on visualization and computer graphics* 26, 1 (2019), 56–65.

[92] Kanit Wongsuphasawat, Daniel Smilkov, James Wexler, Jimbo Wilson, Dandelion Mané, Doug Fritz, Dilip Krishnan, Fernanda B. Viégas, and Martin Wattenberg. 2018. Visualizing dataflow graphs of deep learning models in TensorFlow. *IEEE Transactions on Visualization and Computer Graphics* (2018).

[93] Junru Wu, Yue Wang, Zhenyu Wu, Zhangyang Wang, Ashok Veeraraghavan, and Yingyan Lin. 2018. Deep k-means: Re-training and parameter sharing with harder cluster assignments for compressing deep convolutions. In *International Conference on Machine Learning*. PMLR, 5363–5372.

[94] Xuemei Xie, Xiao Han, Quan Liao, and Guangming Shi. 2017. Visualization and pruning of SSD with the base network VGG16. In *Proceedings of the 2017 International Conference on Deep Learning Technologies*. 90–94.

[95] Xiwei Xuan, Xiaoyu Zhang, Oh-Hyun Kwon, and Kwan-Liu Ma. 2022. VAC-CNN: A visual analytics system for comparative studies of deep convolutional neural networks. *IEEE Transactions on Visualization and Computer Graphics* 28, 6 (2022), 2326–2337.

[96] Marwa Zamzam, Tallal Elshabrawy, and Mohamed Ashour. 2019. Resource management using machine learning in mobile edge computing: A survey. In *2019 Ninth International Conference on Intelligent Computing and Information Systems*. IEEE, 112–117. https://doi.org/10.1109/icicis46948.2019.9014733

[97] Amy X Zhang, Michael Muller, and Dakuo Wang. 2020. How do data science workers collaborate? roles, workflows, and tools. *Proceedings of the ACM on Human-Computer Interaction* 4, CSCW1 (2020), 1–23.

[98] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. 2018. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 6848–6856. https://doi.org/10.1109/cvpr.2018.00716

[99] Tianming Zhao, Yucheng Xie, Yan Wang, Jerry Cheng, Xiaonan Guo, Bin Hu, and Yingying Chen. 2022. A survey of deep learning on mobile devices: Applications, optimizations, challenges, and research opportunities. *Proc. IEEE* 110, 3 (2022), 334–354. https://doi.org/10.1109/jproc.2022.3153408

[100] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223* (2023).

[101] Zhi Zhou, Xu Chen, En Li, Liekang Zeng, Ke Luo, and Junshan Zhang. 2019. Edge intelligence: Paving the last mile of artificial intelligence with edge computing. *Proc. IEEE* 107, 8 (2019), 1738–1762. https://doi.org/10.1109/jproc.2019.2918951

[102] Mingjian Zhu, Kai Han, Enhua Wu, Qiulin Zhang, Ying Nie, Zhenzhong Lan, and Yunhe Wang. 2021. Dynamic resolution network. *Advances in Neural Information Processing Systems* 34 (2021), 27319–27330. arXiv:2106.02898