Model Compression in Practice: Lessons Learned from Practitioners Creating On-device Machine Learning Experiences

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

Mary Beth Kery
Apple
Pittsburgh, PA, USA
mkery@apple.com

Donghao Ren
Apple
Seattle, WA, USA
donghao@apple.com

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

ABSTRACT

On-device machine learning (ML) promises to improve the privacy, responsiveness, and proliferation of new, intelligent user experiences by moving ML computation onto everyday personal devices. However, today's large ML models must be drastically compressed to run efficiently on-device, a hurtle that requires deep, yet currently niche expertise. To engage the broader human-centered ML community in on-device ML experiences, we present the results from an interview study with 30 experts at Apple that specialize in producing efficient models. We compile tacit knowledge that experts have developed through practical experience with model compression across different hardware platforms. Our findings offer pragmatic considerations missing from prior work, covering the design process, trade-offs, and technical strategies that go into creating efficient models. Finally, we distill design recommendations for tooling to help ease the difficulty of this work and bring on-device ML into to more widespread practice.

CCS CONCEPTS

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

KEYWORDS

Efficient machine learning, model compression, on-device machine learning, interview study, interactive systems, design directions

1 INTRODUCTION

Most modern machine learning (ML) models in production today occupy cloud servers of exceedingly more computational capacity and power than the device in your pocket. Yet for delivering intelligent user experiences, there is good reason to move ML models onto personal computing devices people use every day.

On-device ML is the practice of storing, training, and running ML models on an individual's device, such as a smartphone, tablet, or wearable. However, as today's state-of-the-art models grow larger and larger in size (e.g., into the billions of parameters [33, 94, 104, 127]), *efficiency* remains the biggest barrier to on-device ML [9, 23]. An arbitrary ML model placed on a mobile device can easily consume every available resource of the device, whether it be compute, memory, or battery. Creating efficient, on-device models brings new challenges to the ML development process.

In this paper, we argue that research in *efficient machine learning* over the past decade has matured to the degree to where practitioners have an (albeit rapidly evolving) set of techniques to make on-device ML a reality. The key idea is to shrink, optimize, and compress models, while simultaneously maintaining their accuracy. To achieve this, practitioners develop strategies for how to best apply *model compression* techniques to minimize the amount of computational resources needed. Product designers, tool-builders, and ML practitioners today should be considering the benefits of on-device ML for their users:

First, on-device ML can be an enormous win for personal privacy. While standard practice today is to send a user's (encrypted) data over a network to servers for ML inference, on-device ML cuts out this dependency such that a user's personal data never leaves their device. Beyond inference, pre-trained models can be fine-tuned on-device to adapt to an individual user's preferences, while keeping those preferences private and local [10, 49, 54].

Second, models on-device enable intelligent user experiences where they would not be possible otherwise. For example, computational photography models within mobile cameras run inference at high frame rates (e.g., 30 frames per second), which would not be possible if relying on the availability of a distant server.

Third, by going offline, on-device ML can create not only faster but also more portable experiences. Since no data is sent over a network, users can still interact with ML-powered features in situations without internet access or cellular service. This has the potential to broaden access of AI/ML features in more geographic regions, including rural and remote areas [99]. By forgoing a network entirely, ML-powered features can run faster and remain responsive by alleviating network latency in an "offline" mode [23].

Lastly, removing the reliance on servers for ML inference has economic and environmental impact. On-device ML cuts out the cost of a server, which may help individuals, non-profits, or smaller tech firms that previously could not afford server upkeep to now deliver ML-powered features to their users. By reducing society's

reliance on external servers, on-device ML may help reduce the carbon footprint of the cloud [73].

In this paper, we seek to advocate for on-device ML by filling a crucial gap: while efficient ML research and algorithmic advances have progressed tremendously [25, 37, 68, 91, 122, 126], there currently exists little pragmatic guidance [76] in literature, online, in books, or otherwise, for people wanting to create on-device ML user experiences. On-device ML requires both clever algorithmic and user experience design, making this a fundamentally interdisciplinary problem that has received limited attention outside of ML venues. Our work addresses: **RQ: How should a broader audience of HCI and ML practitioners today optimize powerful models to design on-device, ML user experiences?**

To curate pragmatic guidance, we interviewed 30 expert industry practitioners who are uniquely experienced with designing, developing, and deploying on-device ML at scale. We capture the hard-won knowledge, insights, and tricks-of-the-trade that are traditionally omitted from technical ML publications. We draw connections between the design of ML user experiences and the compression strategies experts use to match design goals. We contribute:

- **Tacit knowledge that 30 expert practitioners have developed** around the design processes, trade-offs, and strategies that go into deploying efficient on-device ML. Many decisions around optimization and compression strategy stem directly from user experience and product design.

- **A characterization of the key challenges practitioners face** when creating efficient models. Examples include the tension between optimizing performance against accuracy, and the necessity to work with hardware-level details.

- **Distilled design recommendations for interactive interfaces, tooling, and systems** that show promise to help practitioners optimize models and ultimately proliferate on-device ML experiences.

We conclude by discussing where the broader HCI + AI/ML research, design, and practitioner communities can engage in efficient, on-device ML. We hope the results from this deep dive into a nascent area of ML user experience design helps spotlight its interdisciplinary importance and inspires others to contribute.

2 RELATED WORK IN HUMAN-CENTERED ML AND MODEL OPTIMIZATION

Our work attempts to bridge the Machine Learning, Ubiquitous Computing, Mobile Computing, and Human–Computer Interaction (HCI) communities, to build common language around an ML topic with broad, intersectional impact. Our biggest challenge is in situating this research where no direct prior work exists. In this section, we anchor our work in two main threads of HCI research. First, overlapping with the HCI community, the Ubicomp and Mobile computing communities have long used model compression for on-device ML to make new user experiences possible [24]. Our work complements this literature by consolidating together strategies across many use cases, with the aim of broadening the audience of practitioners who can contribute to creating on-device ML systems (Section 2.1). Second, our research continues a line of HCI work synthesizing lessons from real-world ML

practice [7, 8, 12, 42, 60, 88, 119], and we extend prior work into a previously unconsidered area of ML (Section 2.2). We situate our work in both of these directions, and end by covering related work on model efficiency for on-device ML itself (Section 2.3).

2.1 Ubiquitous Computing: Deploying ML on Mobile & Edge Devices

Challenges of on-device ML often occur in the context of mobile or edge computing [16, 75, 131]. Edge computing refers to small embedded hardware, wearables, or internet-of-things (IoT) devices where at least some computation is performed on-device (the "edges" of a network) rather than by a central server [75]. Example uses of compressed neural networks include facial recognition on mobile devices [129], activity recognition on cameras [71], gesture recognition on smartwatches [117], and respiratory monitoring on phones and smartwatches [15, 20, 61]. Since efficiency is critical for edge devices, the Ubiquitous computing, Mobile computing, and IoT research communities have contributed model compression advances around dynamic models [65, 70, 108], structured sparsity [62], and on-device training [51, 58, 120]. Our work differs from prior literature by not focusing on any single application or technique, and instead synthesizing practitioner strategies for model compression that can be used *across* use cases and hardware types. In our research, we interviewed ML practitioners deploying to a variety of consumer devices, including edge hardware. We distinguish between device type only where advice differs for small embedded hardware, for example in Section 5.4.4.

2.2 Human-Computer Interaction: Studying AI/ML Practitioners

Over the past decade, interview studies from HCI have played an important role in giving the public access to learn from AI/ML work that otherwise happens behind closed doors [7, 8, 12, 42, 60, 88, 119]. Interviews with practitioners are uniquely suited to capture the kinds of process-oriented insight, stories, and tricks-of-the-trade that are traditionally omitted from technical ML publications. For instance, Amershi et al. [7] interview and survey AI/ML practitioners at Microsoft to gather best practices for production ML development. Researchers have studied how AI/ML teams collaborate across diverse technical roles [79, 81, 124], or adopt new technology like AutoML [105–107]. Sambasivan et al. [88] used interviews to profile organizational struggles between balancing modeling work with data quality work. Holstein et al. [42] found key mismatches between academic AI/ML fairness concepts and the kinds of real fairness issues product teams grapple with. Many other aspects of AI fairness, accountability, transparency, and ethics have been found to be practitioner-driven, where design decisions and process have enormous impact [21, 43, 46, 66, 67, 72, 86, 87]. Like this prior work, we examine the production and practice side of on-device ML efficiency to illustrate the connection between ML compression choices and their impact on holistic ML user-experience design.

2.2.1 User Experience Design for AI/ML. Much of what the research community knows today about designing effective user experiences (UX) for AI/ML comes from interviews with seasoned product designers [112, 119, 121, 123]. Leading technology companies have

distilled design wisdom from their own product teams into AI/ML design guidelines as public, educational resources [114]. Examples include Apple's Human Interface Guidelines on Machine Learning [3], Microsoft's Human-AI Guidelines [8], Google's People + AI Guidebook [4], and IBM's Design for AI resources [2]. This body of work emphasizes that designing with machine learning requires new approaches from designers [2, 4, 8, 112, 118, 121]. Our paper contributes to this conversation by illustrating how power and performance issues of moving ML models on-device create real, tangible UX constraints for designers (for example, see Table 2), and offers concrete design strategies for creating effective user experiences around those constraints (for example, see Section 5.4).

2.3 Machine Learning: On-Device ML & Compression Techniques

The literature on efficient machine learning is broader than the scope of this paper, and for a comprehensive look we refer readers to the excellent survey papers cited here and below [17, 19, 22, 68, 101]. For neural networks, Menghani [68] breaks efficiency into 5 areas: (1) compression techniques, (2) learning techniques, (3) automation, (4) efficient architectures, and (5) infrastructure & hardware. Our interviews with practitioners contained more discussion of (1) and (5), and some discussion of (2) and (4). We do not claim that this interview study is a comprehensive look at efficient machine learning. Nonetheless, our work adds a fresh perspective to existing efficient ML literature: instead of detailing specific optimization techniques, here we profile higher-level strategies for how practitioners put techniques into practice to enable user-experience goals.

2.3.1 On-device Inference vs. On-device Training. It is worthwhile to distinguish between on-device *inference* and on-device *training*. On-device *inference* refers to running an ML model on new input to get a prediction. These models are typically pretrained on a server, then delivered to a device. On-device *training* refers to either training a model from scratch or fine-tuning a model on a user's device. While work in on-device ML encompasses both paradigms, in our paper we focus on the (currently) more common use case of on-device inference and leave on-device training for future work. Training on-device is generally considered harder, since training usually requires far more resources [131]. For a survey on the current challenges around on-device learning, see: [25, 64, 75, 131].

2.3.2 On-device Large Language Models and Foundation Models. In recent years, large language models (LLMs) and other generalized foundation models have raised the magnitude of size we expect from ML models [33, 74, 93, 104]. As models get bigger, so do the stakes for model compression and on-device efficiency [13]. Beyond speeding up inference [6], efficiency methods for LLMs also aim to lower their enormous pretraining and fine-tuning costs. Examples include low-rank adaption methods (e.g., LRPD [128] and LoRA [48]), and parameter-efficient fine-tuning methods [26, 27, 30, 59]. The number of empirical works for compressing LLMs from 2023 alone would be difficult to capture, therefore we refer interested readers to the following surveys: [101, 116, 130]. We note that many of the techniques for compression that we discuss in Section 3 are the same key ideas being applied to LLMs and foundation models.

2.3.3 Publicly Available Compression Resources. While most compression techniques originate in academic work [17, 19, 22], many resources for practitioners can be found within web tutorials and ML toolkits. Examples include TensorFlow's quantization-aware training method [96, 97]; PyTorch's experimental support for quantization [83], sparsity [84], and it's accompanying examples [85]; Google's quantization extension to Keras called QKeras [35]; Microsoft's Neural Network Intelligence package and tool [69]; Intel's Neural Compressor library [50]; and Apple's MLX framework [39] and DNIKit [111]. Other examples that target specific hardware include speeding up inference on FPGAs [28] and compressing Core ML models to run on Apple platforms [9]. In terms of other community efforts, the appropriately named TinyML community has published a book [110] and hosts community events and meetups.

3 A PRIMER ON MACHINE LEARNING COMPRESSION TECHNIQUES

To familiarize readers with model optimization, here we give a primer on common ML compression techniques. This background will help ground the study results and provide context for the remainder of the paper. *Note we only cover common techniques mentioned by practitioners in our study—more exist.*

Model compression is a class of techniques used in on-device ML to reduce the computational resources a model consumes. While it is not important to know every detail of every technique, it is useful to understand the variety of techniques at a high-level, and how they can be combined for bigger savings [38]. This overview contains a brief description of each technique, and includes illustrations to build visual intuition. Note that each technique below is truly a family of techniques, each with many nuanced variations. For an in-depth review of the technical descriptions of compression techniques, see the following surveys: [17, 19, 22, 68].

3.1 Quantization

Convert the inputs, outputs, weights, and/or activations of a model from high-precision representations (e.g., fp32) to lower-precision representations (e.g., fp16, int32, int16, int8, and even int2). At a high-level, this coarsens a model. For a detailed survey of quantization-specific techniques and their variations, see [32].

3.2 Palettization (Weight Clustering)

Map the weights of a model to a discrete set of precomputed (or learned) values [18, 115]. 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. In this way, palettization is similar in spirit to algorithmic memoization, a dictionary, or a look-up table. Palettization can make a model smaller but does not make a model faster since it incurs look-up time.

3.3 Pruning (Network Sparsity)

Remove the least-important parts of a neural network to make it smaller [41]. The motivating idea is that modern neural networks are much more dense and overparameterized than is actually needed. Since networks can contain billions of parameters, removing some or many parameters may not impact the final accuracy. Pruning is a large family of techniques [41].

3.3.1 Unstructured Pruning. A model may have neurons or weights that do not much affect a model's decision [32]. In unstructured pruning, the least important neurons or weights can be aggressively removed without affecting model accuracy [41]. Unstructured pruning has been shown to shrink a model by 10–100x its original size [41]. The major downside is that this leads to *sparse matrix operations*, which is a type of math that is slow on most hardware. So although pruned models are small, they may be nearly as slow as the full-sized model [32].

3.3.2 Structured Pruning. Similar to unstructured pruning, the least important neurons or weights are identified, however, instead of removing (or zero-ing out) individual values, structured pruning removes entire structural elements, like channels or filters. Since structured pruning removes much bigger chunks of a model, less pruning will be possible without serious degradation in model accuracy [32]. However, by respecting the structure of the model, these pruned models maintain *dense matrix operations*, which keeps the model fast on most hardware [41].

3.4 Distillation

Train a smaller model to mimic a larger model. Given a larger (teacher) model that is already highly accurate, transfer its learned features to a smaller (student) model by having the smaller model replicate the larger model's output. One can then apply other compression techniques to the student model for further optimization [32, 82]. For a survey of distillation techniques, see [36].

3.5 Efficient Neural Architectures

Some model architectures are specifically designed to be small and efficient. Prominent example architectures include MobileNets [47, 89], ShuffleNets [125], and EfficientNets [95]. All of these examples are designed to improve the efficiency of convolutions, which are vital to fitting computer vision models on-device [19]. One can use other compression technqiues on efficient architectures to create even smaller models [19].

3.6 Dynamic Models

When thinking about ML applications, most people think of a single model with fixed inputs and outputs. Dynamic models differ in that they take into account the differing prediction difficulty of each data input. Depending on the input, a dynamic model may adapt its preprocessing steps, computational path, or model choice all together [132]. While many types of dynamic models exist, we only discuss a selection of techniques that are mentioned later on.

3.6.1 Early Exit Models. Allow for completing a prediction without the need to pass through the full model. Motivated by the fact that some data points are easier to predict than others, early exit models run intermediate feature representations through additional output layers to check if the prediction is sufficiently correct. For example, if a model is already confident about the current prediction, it finishes early rather than continuing to the next prediction layer.

3.6.2 Gated Models. Train a smaller, approximate model whose prediction decides whether or not to invoke a larger model. Similar to early exit models, some data points are easier to predict than others. A fast, smaller model gates a larger model, for example by again using the confidence of the initial prediction. At a high-level, these are systems of models that decide whether or not to spend extra compute if a prediction is still uncertain or unknown.

4 INTERVIEW STUDY METHODOLOGY

4.1 Study Protocol

To capture emerging practices around efficient machine learning, we conducted semi-structured interviews [14, 53] with 30 ML practitioners at Apple to study how they approach model compression in their own work. Our interview questions are outlined in Appendix A. We gave participants ample time to flexibly speak to their specific work and express other topics beyond our question set that they felt were important to effectively optimizing models [53]. The interviews took place between March and July 2022 with each conversation lasting from 45 minutes to 1 hour. For all interviews, one researcher lead the interview questions, while another took detailed notes. Where a participant approved, we also recorded conversations to refer back to during analysis. No compensation was given, since all participants were salaried employees. At the end of the study, we briefed participants and their teams on our results. Our study protocol was approved by an internal IRB.

4.2 Participants and Recruitment

As discussed in Section 2, in-depth interview studies with ML practitioners are uniquely suited to capture experts' tacit knowledge for the purpose of generating new, publicly-available guidance [7, 8, 12, 42, 60, 88, 119]. Our organization has a rare concentration of efficient machine learning experts, so we first reached out to known individuals. We then used a snowball sampling strategy to reach a broader network of people involved in efficient machine learning. To balance different perspectives, we sought practitioners working on ML for different domains and tasks (e.g., vision, language, and sensing). As the study evolved, we also chose new participants to help fill-in our largest areas of uncertainty. Our participants, listed in Table 1, include ML research scientists, engineers, and managers spanning a wide breadth of application

domains. A natural saturation occurred when participants began to repeat know-how we had already recorded and began to *only* suggest participants we had already included.

4.3 Qualitative Data Analysis

The first two authors conducted an iterative thematic analysis method to group common motivations, challenges, and best practices of model compression into categories [34]. As we conducted more interviews, we continually discussed and updated these categories to reflect our new findings. Each participant's data and transcripts were independently reviewed and manually coded by the first two authors at least twice using inductive coding [98]. In total, we spent 30+ hours interviewing participants and collected 23,500+ words of additional notes outside of the audio transcripts. The final categories were split into two main sections: Section 5 "Study Results" and Section 6 "Design Opportunities for Efficient ML Tools and Interfaces." The composition of these two sections was formed by ordering major categories and hierarchically grouping within categories when relevant.

4.4 Methodological Limitations

While we found that learning from expert practitioners was tremendously valuable, any interview study has limits for generalizability. Interview studies suffer from smaller population samples than other methods like a survey. We chose to hold in-depth exploratory conversations with each participant—which a survey cannot support as well [53]. On the other hand, we did not directly observe participants conducting their efficient machine learning work, which limits the specificity of our findings. It was not possible to observe all participant's activity or model artifacts due to the sensitivity of their work. Another concern is bias from a participant's role in the domain [14], as well as the power dynamics between interviewer and interviewees [56]. This study was conducted solely within one organization, therefore practitioners may hold organization-specific beliefs and practices [90]. Despite our conscious efforts to recruit across ML application domains, we noticed a skew towards vision-based applications on images, video, and 3D scene data. While interviews were conducted prior to the public rise of LLMs in late 2022 [74], many participants had NLP optimization experience, and as discussed in Section 2.3.2, the same overall approaches for efficient machine learning are currently being applied to LLMs and other foundation model modalities.

Taking these limitations together, we caution readers to not consider the advice we detail from participants universally generalizable. However it is with confidence that we share the rich pragmatic guidance these 30 experts have to offer.

5 STUDY RESULTS

To answer our primary research question **RQ: How should a broader audience of HCI and ML practitioners today optimize powerful models to design on-device, ML user experiences?**, we first profile *who* is working in model compression today, as emerged from our participant pool (Section 5.1), and give a high-level state of efficient machine learning in practice (Section 5.2). We then break down our results along a typical AI/ML development process [1, 5, 29, 113], and use the machine learning workflow outlined

by Amershi et al. [7] as a reference point to map onto (in summary: *model and data requirements*, *model development and training*, and *model evaluation*). Similar to specifying *model requirements* within ML [7], we characterize how practitioners design on-device machine learning experiences (Section 5.3) and plan model budgets (Section 5.4). Next, we discuss how compression affects the *model development and training* process, and offer considerations for how people can strategize applying compression to their own work (Section 5.5). Lastly, we describe how practitioners *evaluate compressed models* to balance accuracy versus performance—and avoid accidental compression artifacts (Section 5.6).

Throughout the results we highlight actionable takeaways in call-out boxes . Note that some of these strategies are unique to model compression for on-device ML, while other strategies add a model compression twist to already-familiar software efficiency ideas. We include both to balance domain-specific novelty with the key advice for on-device ML.

5.1 Participants and Emergent Personas

Who does on-device ML efficiency? Of the 30 people we interviewed, participants had an average of 7.1 (max 12) years experience with ML, and an average 4.1 (max 10) years experience with efficient ML. Our participants had diverse breadth of expertise across domains, detailed in Table 1. Rather than their job title, we found that participant perspectives on efficient machine learning aligned more closely with their application focus. For the purpose of understanding practitioners contributions, we define three distinct, emergent personas that best describe our participants:

- **Compression Experts (E1–E13 in Table 1):** Seasoned research scientists and experienced engineers that lead on-device ML initiatives, develop new compression techniques, and consult on machine learning optimization efforts. Given the amount of experience these people have, they often manage or lead teams. *Example: An ML research scientist who has a PhD in model compression and is developing novel techniques for model optimization.*

- **Machine Learning Practitioners (P1–P11 in Table 1):** ML engineers and data scientists that build and deploy models on-device where certain optimization budgets must be met. These people use compression as a means to an end rather than solely studying efficient machine learning. *Example: An ML engineer optimizing a model's size to shrink it to 1MB while maintaining high accuracy.*

- **Tooling Engineers (T1–T6 in Table 1):** Engineers and developers that focus on building frameworks, infrastructure, and tools for efficient machine learning. *Example: A software engineer building and maintaining an organization-wide tool to help others compute efficiency metrics.*

Comparing the personas, we see clear differences between applications. For example, compression experts are heavily (and nearly exclusively) focused on research, ML practitioners work across the most diverse set of domains (e.g., vision, NLP, sensing, multi-modal models, fairness, and hardware), and tooling engineers focus on internal tools and compression algorithm implementation. Throughout the paper, we label representative quotes from participants by

Table 1: A summary of the 30 participants from the interview study. Participants are grouped by three emergent personas: Compression Experts (E), ML Practitioners (P), and Toolkit Engineers (T). Participants indicated the number of years they have spent working on machine learning (light blue squares), and of those years how many have been spent working on model compression and optimization specifically (dark blue squares).

ID	Experience (Compression / ML Years)	Job Title (Manager ✓)	ML Application
Compression Expert			
E1	10/12	ML Manager (✓)	Efficient ML research
E2	7/12	ML Manager (✓)	3D computer vision
E3	5/12	ML Manager (✓)	Efficient ML research
E4	5/10	ML Engineer (✓)	3D computer vision
E5	9/9	Research Scientist	Efficient ML research
E6	6/8	ML Engineer (✓)	Efficient computer vision
E7	5/8	ML Engineer (✓)	3D computer vision
E8	2/8	ML Manager (✓)	Efficient ML research
E9	5/6	ML Engineer (✓)	Efficient computer vision
E10	5/6	Research Scientist	Efficient ML research
E11	5/5	ML Engineer	Efficient ML research
E12	5/5	Research Scientist	3D computer vision
E13	3/5	ML Engineer	3D computer vision
ML Practitioner			
P1	6/10	ML Manager (✓)	Efficient computer vision
P2	4/10	ML Engineer	Machine translation
P3	3/10	Research Scientist (✓)	ML sensing
P4	4/9	ML Engineer	Machine translation
P5	5/8	Software Engineer	Computer vision
P6	1/6	ML Engineer	Multi-modal ML
P7	1/6	ML Engineer	Multi-modal ML
P8	2/5	ML Engineer	ML fairness
P9	3/4	ML Engineer	ML hardware
P10	3/4	ML Engineer	3D computer vision
P11	1/4	Software Engineer (✓)	Multi-modal ML
Tooling Engineer			
T1	5/10	ML Manager (✓)	Efficient ML tooling
T2	3/6	Software Engineer	Efficient ML tooling
T3	2/6	Software Engineer	ML fairness
T4	4/4	Software Engineer	Efficient ML algorithms
T5	3/3	Software Engineer	Efficient ML algorithms
T6	2/2	Software Engineer	Efficient ML tooling

their personas to illustrate the main findings from the study. The persona labels offer additional context for situating a participant's background and perspective into the larger discussion.

5.2 The State of Efficient ML in Practice

Creating an efficient machine learning model is a *"large design and constrained optimization problem"* [E5], where practitioners have to weigh decisions between many model performance metrics, such as model storage size, inference latency, power usage, device temperature, and model behavioral metrics such accuracy, precision,

recall, and others. To many readers, this characterization may sound like a job for an automated optimization algorithm. However, a crucial finding of our results is that automation is not yet possible, do to the degree of ML, product design, and human expertise that goes into creating on-device applications. Experts emphasized there is no single solution, *"silver bullet"* [E12], *"turnkey solution"* [E5], or *"golden recipe"* [E7] for successfully building efficient machine learning models. Partially, this is due to the rapidly moving-target of new state-of-the-art model architectures and new ML applications that may require a custom approach.

Figure 1: Practitioners start with a feasibility model (A), which is a model of any size that demonstrates that the ML task works. Next, an architecture search looks for a smaller model that is equivalent to (A) and suitable for devices (B). A team decides a budget based on how much more efficient model (B) must be to deploy. Practitioners use techniques to compress their model (B) to reach the budget (C).

> *"[Compression] works super well sometimes, but sometimes totally fails. Each project is too dependent, recipes tend to only work in some situations."* — P2, P6, P7

The expertise for creating efficient ML models is currently held by a relatively small subset of ML practitioners, where *"knowledge is handed-down from the few who know how to do it successfully,"* [E3]. These people are referred to as artisans and *"based on earlier experience, know it's possible"* [P10] to create highly efficient models. Reducing a model's size by half, or by an order of magnitude, takes additional time and considerations [E7]. *"It's like black magic or the dark arts, but you will get better with time,"* [E7]. Readers may recognize the "dark arts" or "artisan" language from earlier days of contemporary ML [40, 80], before any real widespread knowledge base was established. We hope these study results support growing such a knowledge-base for efficient machine learning, and it's role within a holistic ML process:

> *"[Efficient ML] means a lot of things to different people. Specific techniques on models weights help reduce size, but to get an efficient model comes from more careful design of the loss function, the system, which parts should and should not be modeled with ML."* — E5

In practice, building efficient ML models describes the process of either building a new model from scratch or modifying an existing model to shrink it enough to fit performance constraints, such as model size, latency, power consumption, and even heat [P3]. Newcomers often gravitate towards the latest model compression techniques from literature as a first-step, while experts who consult on these projects encourage taking a step back to consider the entire data and model lifecycle: *"Don't do [model compression] blindly. Don't do [model compression] in a rush,"* [T1]. *"Philosophically, [you] need to look at whole problem, not as an afterthought,"* [E3].

In addition to barriers of overall experience, efficient ML work differs perhaps most significantly in how much it requires a deep understanding of hardware details. Low-level hardware details are not something that typical ML or software engineering usually needs to consider. *"[ML engineers] feel a bit intimated, and people don't feel like they understand this stuff well,"* [T2].

5.3 The Metrics of Efficient, On-Device Models

For many practitioners we spoke to, efficient machine learning is of immediate importance to their work in developing new product features. Efficiency is a deciding point on which features get approved to ship to users, so efficient ML enables user experiences that otherwise simply could not exist [T5, P3, E12, P2, E9].

> *"Who cares if we can detect X if your model takes too much space? It will never make it to engineering requirements."* — P3

As an umbrella term, *efficiency* encompasses many metrics. Common metrics that practitioners linked to user experience are summarized in Table 2. Some may be familiar with metrics commonly considered in ML workflows, such as latency and model storage size. Other metrics may come as more foreign to an ML engineer accustomed to deploying their model server-side, e.g., device temperature. When a user is holding and carrying around a device all day, metrics like heat and battery life impact are crucial.

The prioritization of efficiency metrics depends on the model type and its intended usage. While different practitioners we spoke to were focused on optimizing different aspects of efficiency, they all had a shared concept of a *model budget.*

5.4 Deciding Model Budget

A model budget encompasses thresholds for speed, accuracy, size, or the amount of any specific resource a model is allowed to consume. Model budgets are created individually per model, and are based as much in product and UX design as they are in device constraints.

5.4.1 Budgeting by Technical Feasibility. For new and novel ML applications, it is simply unknown on the onset how computational intensive a model might be. Practitioners refer to reported metrics from related ML research and an organization's other models to sketch out an initial budget:

> *"At the beginning, [we do a] 'back of envelope calculation' where things need to be as honest as possible to what's realistic. At the beginning, you're more focused on accuracy. Over time there is refinement."* — E7

A common refinement practice is outlined in Figure 1. First, ML engineers will work on a "feasibility" model simply to see if the ML task *"is possible at all"* [P3]. Once the feasibility model works at sufficient accuracy (Figure 1A), engineers work on an architecture search to find a smaller, more efficient model that achieves the accuracy goal (Figure 1B) [P3, E7]. Next, the budget is refined based on the potential to shrink the model using compression: *"given [an] accuracy goal, what is the biggest percentage reduction possible?"* [E11]. Some teams do this reduction estimation on their own, while others bring in ML compression experts [T5]. Since the compressed-model budget (Figure 1C) is often estimated *before* engineers embark on model compression, the budget remains open to refinement, subject to product design constraints. Ultimately, if an ML-powered feature requires a strict model budget, the model will not ship until engineers have found a way to reach those thresholds [P3].

5.4.2 Budgeting by User's Experience of a Model. Many aspects of budgeting come directly out of product feature design for where, when, and how often a model will be running.

⊞ *One-Time, Real-Time, or All the Time?* Latency budget is typically a per-feature UX decision dependent upon how a user perceives the model output and how often the model needs to run.

Table 2: Key aspects of on-device ML efficiency that impact user experience.

User Perceivable Metrics	Negative User Impact
📦 Model storage size	Modern deep learning models can easily grow to gigabytes. Devices have finite storage, so a model must not occupy so much space that it interferes with a user storing their own content.
🔋 Power usage	Users expect their devices to have long battery lives, but a resource-intensive model can quickly drain a battery. Addressing battery drain is particularly challenging for older devices with lower battery capacity and health.
🔥 Device temperature	ML models that consume lots of computational power at once can heat up a user's device to where it is uncomfortably warm to the touch. Heat can also trigger thermal throttling, where the device slows down to avoid over-heating.
🕐 Inference latency	Taking a long time to run inference can be frustrating for a user waiting on the result, and can make the experience feel unresponsive.
🎞️ FPS (frames per second)	A special case of inference latency. When processing live inputs (e.g., video), this latency can create delays that makes the output appear choppy or unresponsive.
All metrics: model tested during high device resource load	Resources are shared. A user's device may be running many things simultaneously, including multiple ML models. A single model that demands too many resources will slow everything down.

For example, in a photography app when tagging people in a new photo, a model needs to work fast enough so that users do not notice a delay. Models on-device remove the wait on network response times, which make these latency budgets easier to achieve [P2].

Models that only need to run once generally have a looser resource budget than continuous models. For models running in real-time, only a few milliseconds per inference may be the maximum budget [P1]. This is most crucial in real-time scenarios where the user can perceive the model working, such as in live video applications [E13]. A user will notice if their video feed appears choppy or a background filter appears delayed. For this reason, latency budgets need to be as low as possible. Similarly, strict latency budgets are given to models that need to run all the time, i.e., "always-on". Always-on models are continuously using the device's resources, so they are required to be as quick and lightweight as possible. Some always-on models require a timely response to the user, for example, an always-on model that listens for a user to trigger a voice-assistant.

🕐 *User Opt-In or Background Task?* There are some applications where ML is the primary enabling technology, such as voice assistants or language translation apps. If a user explicitly takes action to trigger a model, the user is in control of when the model runs (or does not run). Product designers will allow models a larger budget for power and compute resources when a user explicitly opts-in [E11, E7, P2]. It is also common to have supporting models running in the background. Similar to any other background task, these models require a strict budget to stay unobtrusive:

> *"If the user will never perceive the model, we still want the memory footprint to be inconspicuous so it doesn't interfere with them using their device."* — E7

🛏️ *Can it Wait for a User to Sleep?* Many devices today will download and install routine updates at night, or whenever the device's user sleeps and has their device connected to power. Updates are intensive computational tasks that can be disruptive to normal device function, thus it is conventional to wait until a user is not actively using their device to initiate. By the same logic, scheduling a model to run overnight is a good option if the model's output is not something needed immediately. If a user's device is plugged in, this also alleviates the concern of a model draining battery. Disruption to the user is minimized, so models running overnight are also free to take a longer time with a bigger resource budget [E13, P3].

> **Strategy #1**: First estimate the accuracy and latency of the original model architecture you want to run on-device. Then estimate how often the model needs to run (less often is "easier") and what it needs to deliver (less accurate is "easier") to budget the *worst-case* performance that will be acceptable for your user experience. The gap between your starting estimate and worst-case budget will tell you how feasible your on-device ML plan is.

5.4.3 Top-down Budget by Application & Device. As illustrated in Figure 2, an application or device has finite resources—and ML models are only one component of a system. For this reason, major budget allocation is typically decided by people with far reaching views and ownership in an organization [E1, P1, T1, P9]:

> "[The product lead] *looks at budget for whole feature and allocates budget to individual models from there. There are dozens of algorithms.* [Each model] *gets budget based on priority, practicality, and executive decisions on what is the most important algorithms to give space/time to.*" — T1

The difficulty of hitting budget can vary enormously depending on the model. Thus, negotiation over budget allocations can continue and evolve until late in development [P1].
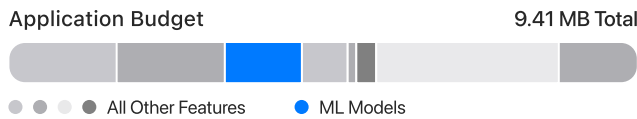
Application Budget   9.41 MB Total

● ● ● ● ● All Other Features   ● ML Models

Figure 2: Models needs to share space with the rest of an application. Model budget typically reflects how "valuable" a model is compared to all other features of the application. The above hypothetical example shows how much budget a model (blue) has in the overall system.

🧊 *Size Budget.* Any one application should not take up too much disk space on a device. The challenge is that as more and more new features make use of ML, there are more models to fit in the same amount of storage:

> *"Because of* [neural networks]*, and the rising popularity of huge transformer models, compression becomes more and more important."* — E9

🔋 *Power Budget.* A model's power budget is a measure of how much battery it is permitted to use. Power consumption of a model can be tricky to estimate, and should be measured empirically on the target device for the most reliable estimate [E6, T4, P9]. Once again, real-time or always-on models require much stricter power budgets, because they continuously draw power. Instead of measuring power at one moment of execution, these continuous models can be thought of in terms of the total percentage or total minutes of battery they consume in a day [E6, P3, E7].

🔥 *Heat Budget.* Closely related to power consumption is device temperature. In the words of E7: "*ship high-quality ML models that don't melt the user's device.*" By melting, E7 refers to both a device that is uncomfortably warm to the touch, and a device that has been forced into thermal mitigation. Thermal mitigation, or thermal throttling, is when a device slows down processing to protect itself from overheating. Hot and slow devices makes for a terrible user experience that should be avoided. Thus, memory and power budgets must be set according to thermal measurements [E6, E7, E4, E1, E2]: "*Heat throttling is everyone's budget!*" [E4].

Multiple Models in Concert. A development team may measure the power, latency, or memory load of a single model as they develop it, but testing a single model in isolation is insufficient in situations where multiple models are powering an app or experience at the same time [E6, E13, E10]. Multiple models running simultaneously will affect the overall memory pressure experienced by device at any moment, and thus the latency of each model as well as total power consumption. For these situations, teams need to experimentally test models running in concert to get accurate measures of total consumption, and experimentally refine their budget for each individual model from there [E13, E10]:

> *"You could use too much memory, and that's a no-go. You don't know that until you test on device. [The memory bound] is not something you can compute ahead. There are multiple models happening at once, so until they are all put together you can't see exactly what the memory is going to be on a process."* — E13

Strategy #2: Budget a model's resources by it's value-add to your overall experience. For a reliable estimate, run your model on the actual target devices early and often during model optimization. Measure compute usage, memory pressure, battery consumption, heat, and your model's storage size relative to the overall application. Run the model alone and in realistic, high-load scenarios where many process are running simultaneously to find out if your model is too resource greedy. Remember to not be a bad citizen: an ML feature that does not respect the resource needs of the overall device will result in a poor user experience.

5.4.4 Edge, IoT, and Low-power Devices. Devices like laptops, tablets, and mobile phones have far more memory and power resources than smaller, low-power "edge" devices, such as wearables or IoT devices. For edge devices, budget is tight enough that even the bytes of the modeling code text itself can matter [E11]. For these devices, an expert recommended avoiding neural networks all together due to their high cost, and approximate the same accuracy as much as possible using the lowest profile conventional machine learning algorithms, such as decision trees [E11].

5.4.5 Gating & Variable-Budget Options. There may be features where it is not (yet) technically possible to fit the ML component entirely on device. One option is gating, where only *some* inference is done on-device: a small simplified model lives on-device and is only highly accurate for common inputs. If the small model detects complex or uncertain inputs, it invokes a high-powered ML model that uses more resources, or it sends the harder input to an external model on a server (see background Section 3.6.2) [E11].

Another option is variable budgeting. Product teams may be able to keep essential parts of a feature running smoothly under intermittent network availability by dynamically changing whether it uses higher-accuracy models or lower-accuracy models. Similarly, product teams can also choose to lower ML accuracy when needed to keep a feature running under high-device memory load:

> *"So people think about model compression as static resources, but the actual resources on device are dynamic based on what's running. So it's useful to on-the-fly change how much resources your model uses."* — E10

For any model, there will be some size and efficiency budget at which the model can be shrunk without any noticeable difference to the user. Below that threshold, accuracy degradation may negatively impact UX. A budget must balance the best model accuracy with the lowest possible resource footprint.

Strategy #3: On-device ML does not need to be all-or-nothing. If current compression techniques cannot fit your ML feature on-device, try breaking the feature down into subtasks. Delegate smaller subtask models to live on-device and delegate larger ML workloads to a server if appropriate. Prioritize on-device ML for feature subtasks that will help preserve your user's privacy and keep critical functionality responsive in the absence of a network connection.

5.5 Applying Compression during Development

Participants strongly emphasized that applying model compression *is an investment*. Compression techniques can take intensive engineering effort to apply—and may fail. There are many circumstances that can cause one compression technique to generate enormous savings in one model, and completely fail to produce any savings in a different model, potentially after weeks of wasted engineering effort. While this work is not an exact science, we detail general tips and guidance from practitioners. Note that most of these techniques assume the model to compress is a neural network.

5.5.1 Maximize Architecture Savings First. As shown in Figure 1, practitioners often start optimizing a model by finding a smaller architecture first. *"If you're running a common ResNet, look at strides, layer widths, and the number of layers"* [T5]. Participants also recommended neural architectures designed specifically for device efficiency [E13, P5, E3, E11, T6], such as MobileNet [47]. Some applications do not need a heavy neural network; instead it might be possible, and even beneficial, to convert into a simpler decision tree or SVM [P3]. Architecture savings are seen as a more reliable first approach before attempting compression [E13, E1]:

> *"If you're trying to reduce the size, model compression is effective, but more like a last resort. If you have a model that is just overkill, [compression] can shave off 20-40%, but just changing architecture to something smaller will have way bigger impact."* — E11

Due to the high training cost of some models, participants discussed manual architecture selection (as opposed to auto-ML solutions that automate various stages of ML development) based on their expertise: *"It's really trial and error by hand monitoring the hardware usage [and] identifying bottlenecks,"* [E2].

5.5.2 Check the Architecture Against the Hardware. A critical caveat to architecture optimization is that it is easy to be fooled. A common mistake is to trust in the reproducibility of academic results: *"Don't expect what is "efficient" in a paper to exactly work in practice,"* [E7]. The issue is that at the lowest level of computation, different hardware optimizes for different operations. Reported efficiency measures are only reliable for the specific hardware set they were run on. Even for specialized accelerator hardware designed for ML, the hardware will likely only support certain types of operations. Since hardware cannot be altered at the same pace as software, the latest neural layer architectures from papers may not be possible to run efficiently on-device. Practitioners emphasized the need to adapt their neural architecture to better fit hardware [T5, E7, P9].

5.5.3 Test Against a Range of Hardware. In conventional machine learning work, and more generally much of modern software engineering, it is not required to know the details of the hardware that code will run on. In efficient machine learning, hardware may be one of the greatest challenges when working in on-device ML:

> *"You must consider different layers of abstraction of a model from code to hardware. This can be hard for developers to understand."* — E6

To further complicate matters, often practitioners are not considering a single specific hardware implementation, but rather a class of hardware. Classes of hardware include platforms such as mobile phones, smartwatches, tablets, computers, and others. Consider a practitioner building a model for a smartphone. There are many types of smartphones, and versions of existing smartphones each with their own memory and compute details. Hardware also changes over time, and practitioners may need to consider older versions of hardware that have already been shipped. In the same way that front-end developers build responsive UIs and applications for multiple screensizes, so too do ML practitioners now need to optimize and test multiple hardware implementations specific to a set of devices [E6].

> **Strategy #4**: If starting from scratch, chose a model architecture specifically designed for mobile devices, but be aware that an architecture may not perform as-advertised due to implementation differences in hardware. It is crucial to profile models on your physical target hardware—and for every hardware version you aim to support. Older devices usually have fewer compute resources for your model.

5.5.4 Determine if a Model is Memory Bound or Compute Bound. Another useful strategy is to consider ahead which metrics are likely to be the biggest issue for a model type [T5]. A model is *memory bound* if the size of the model or the size of the data is the biggest issue. A model is *compute bound* if the speed/latency of the model is the biggest issue. For instance, *"Computer vision tends to be memory bound and not compute bound. It's fast but the data sizes are massive"* [T5]. Participants also suggest going layer-by-layer to identify bottlenecks since individual layers can be either memory or compute bound individually [E2].

If memory-bound, architecture changes, sparsity techniques, or palettization techniques can help reduce model size. On-disk size is considered the easiest memory savings to get from model compression [E13]. If data transfer size is causing the memory issue, reducing data resolution for model processing can help, e.g., reducing video data from 1080p to 720p [E8]. If compute-bound, techniques like quantization can help speed up and simplify complex matrix math. Quantization can also save memory since smaller numbers help keep computation in cache memory [E12]. Cache locality is a common latency bottleneck to check [E6, E12], that also helps with power problems:

> *"Latency is easier than power to improve. You can check cache locality, moving data takes a lot of power so that's a big one to check."* — [E6]

5.5.5 Accuracy v. Effort Trade-Off. Practitioners we spoke with had a sense of *"accuracy v. effort"* [E6] or *"risk v. reward"* [E13] for some of the most common compression approaches. As a general heuristic, compression techniques that need little-to-no retraining will be cheaper to apply—but at the cost of steeper model accuracy degradation. Since some of these models take days, or more, to train, involving complex compression logic in the training process is costly. Practitioners must consider cost in money, time, and effort that implementing a specific compression technique could take.

($) *Post-training Quantization.* Quantizing a model's weights *after* the model has been trained was widely considered the first go-to compression technique for many applications.

"So quantization is a big one. You can usually quantize to 8-bit integers without losing accuracy. This isn't always the case. You do need to be careful where you apply quantization. But generally it's a pretty generic technique across different hardware. You can usually cut from `fp32` *to* `fp16` *and can get speed up by going to integers."* — E4

Since weight quantization can be done without additional training, it is considered cheap. 10/30 participants discussed using this approach [E6, P8, E9, T3, T5, T1, E13, E7, T2, T2]. Although post-training quantization is considered *"easy"* [E9] as far as ML compression techniques go, practitioners emphasized that it still often takes complex code to implement and there are many algorithm variations [32] to experiment with [T5]. For models that need high accuracy, post-training quantization may not be enough to hit budget without unacceptable accuracy degradation [E9, E4, E13, E5].

($$) *Compression with Fine Tuning.* When model accuracy goes down after quantization or pruning, fine tuning can help recover the loss [32, 41]. Fine tuning a model costs training time, but is necessary in many situations to recover accuracy. Pruning techniques almost always require fine-tuning on the pruned model [41]. 6/30 participants discussed successfully utilizing this approach [P8, T3, T1, E8, E13, E10].

($$$+) *Compression-aware Training.* Though it introduces additional complexity to model training, compression can be approached as yet another optimization that a model needs to learn during training. 11/30 participants discussed having experience with quantization learned during training [E6, E9, T5, E3, E5, E7, E10, E1, E12, T2, P10]. This approach is generally considered best-in-class and often required when designing always-on ML experiences that require low latency, e.g., real-time computational photography models. Practitioners recommended training-aware compression when a model needs to be compressed *significantly* to meet budget while keeping high accuracy:

"If you want to go to lower bit quantization, such as 4 or below, it's almost impossible to use post-training quantization because the difference in accuracy gets way too big. So for this level of compression you need to do training-aware compression." — E9

Some practitioners had less experience with learned sparsity techniques and called them *"high risk, high reward"* [E13] because they are much harder to control compared to post-training sparsity techniques. Although training-aware compression is considered the best form of optimization [32], a major drawback is that is must be included in initial model training: *"Not starting early with compression is a dead end,"* [E3].

Practitioners use their experience and early testing to judge how much compression a model will need. For example, large vision models may start far exceeding their size and power budgets, and require heavy compression to meet budget. In these scenarios, training-aware compression is likely necessary, which requires planning ahead: *"It has to be done from day 0, not day 100,"* [E7].

For other applications, practitioners suggest estimating how much compression will be feasible with simple post-training quantization. To estimate quantization savings *before training a model,*

first initialize the ML model architecture with random weights, then quantize, and test the model's speed and size on-device. Even a coarse estimate may be worth getting a sense of the magnitude of savings from quantization [E13]:

"[It] gives you a sense of how much quantization is going to help at all before you go through expensive training process." — T5

If post-training techniques produce results that are too far from budget goals, teams can then pivot to training-aware approaches.

Strategy #5: Before you heavily invest in a particular compression technique, start with a cheap estimate of how much savings you might expect to gain. For example, initialize a mobile-friendly architecture with random weights, then quantize it. Profile this "stand-in" compressed model on-device and compare its efficiency to your budget. If it meets budget, then quantization may be enough. If it is far off budget (such is the case with many real-time computer vision models), consider more intensive training-aware compression techniques.

5.6 Evaluating Compressed Models: Efficiency, Accuracy, and Artifacts

The goal of model compression is to reduce a model's size while preserving its accuracy, or what some refer to as *"acceptable accuracy degradation"* [E8]. In general, past a certain point, shrinking a model will likely degrade its accuracy. Conversely, in literature there are examples showing that compression can actually improve accuracy, by acting as a model regularizer and forcing the model to generalize better [55, 100]. This phenomenon is sometimes referred to as Occam's Hill [41]: as light compression is applied, there can be an slight accuracy bump as the model is forced to generalize; however, as one increases the compression strength the model becomes *too general* to properly work and accuracy quickly drops. While these results have been observed on academic benchmarks, practitioners said that their models tend to generalize quite well already, so most often they see little improvement from the regularizer effect of compression [E8, E1]. Thus, we focus this discussion on the much more common scenario where practitioners must wrestle with accuracy degradation.

5.6.1 The Trade-off Curve Visualization. To empirically compare multiple compressed models and select the one that satisfies budget requirements, practitioners typically plot a model behavioral metric (e.g., accuracy) on the y-axis against any performance metric (e.g., model size, model latency, or power consumption) on the x-axis [102, 103]. This is illustrated in Figure 3. Each dot represents a model architecture and compression pair. These charts are called multiple names throughout different teams, including the trade-off curve, the accuracy Pareto curve, or the tuning curve.

P3 described a scenario when optimizing over 6,000 small models. In their application, the ML model was already on the order of kilobytes; therefore, they could retrain many new models quickly. Comparing thousands of models in parallel, they plotted the F1-score on the y-axis against the model size on the x-axis. This arrangement formed the usual curve. Teams then filtered out models that do not

Accuracy (%)

Latency (ms)
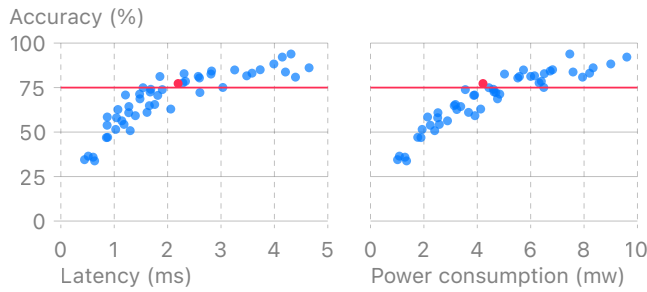
Power consumption (mw)

Figure 3: A common chart used for model comparison and selection, where the y-axis is a model behavioral metric (e.g., accuracy) and the x-axis is any number of performance metrics (e.g., latency or power consumption). Each blue dot represents one model trained with different architectures or compression schemes, and the red line indicates the accuracy budget threshold. Practitioners look for the "knee in the curve" across charts: choosing a model above the accuracy threshold that minimizes latency and power consumption. In these charts, the selected model is colored red.

satisfy their budgets, and ultimately selected a single model in the "knee" of the curve that has the best balance between what metrics they care about (e.g., Pareto optimal).

5.6.2 Compounding Changes Degrade Accuracy. A common challenge with maintaining accuracy is that model compression techniques can compound in unintended ways. In some scenarios, small optimizations throughout a model build on one another, so by the time a data input reaches the end of a network, its prediction is totally off. In ML, this phenomenon is similar to the concept of exploding/vanishing gradients [11, 78]. The effect of this compounding error is not obvious to spot in beginning [E9]. For instance:

> *"When you have addition that takes two quantized values to a quantized output, it's not easy to check that the range of the inputs are the same as the output ranges. You tend to lose resolution in the output where one branch dominates the range, and you lose the range of the lesser branch. You need additional care to check that they have the same range tracking during the forward quantization pass."* — E6

For quantization, practitioners advise a "gut check" to ensure the quantized weights and activations match the ranges of the original model, or else accuracy will degrade [E6].

5.6.3 Robust, End-to-end Data Evaluation. Since the amount of success in model optimization varies significantly by architecture and task, *"without a clear evaluation strategy you won't know if you're making things better or worse,"* [E11]. Some ML-powered experiences are composed of multiple models working together. Participants said they first test models individually to ensure they work standalone, then evaluate the multi-model systems to test compound effects [E10]. Practitioners said the curve visualizations (Figure 3) are often the output of evaluation pipelines.

Many teams do extensive model evaluation, error, and failure analysis [T2, E12, P11, P6, P7, E10, T4]. *"We cannot assume compression doesn't change model behavior, so we look at confusion matrices and instances where the model gets it wrong"* [E12]. In one project that P11, P6, and P7 worked on where small mistakes could lead to a poor user experience, they built "unit tests" for different curated testing sets to monitor accuracy over time. E6 also emphasized using data unit tests to observe how single instances move through a model to both understand how much computational change occurs after compression and make the necessary adjustments to the model code. *"Data so influential to hitting accuracy targets"* [E3].

Strategy #6: Compression can degrade the accuracy of a model and change its behavior in unpredictable ways. It is essential to create a robust evaluation pipeline (e.g., defining metrics, curating test sets) before you start optimizing your model, so that you can reliably observe shifts in model error afterwards. To prevent degradation from a failed optimization, compare optimized models with varying amounts of compression to your original model, inspecting the metrics, subpopulation behaviors, and internals, such as weights and activations, to ensure they are within expected ranges.

5.6.4 Model Compression Artifacts. As with other types of media, such as image, audio, and video data, if too much compression is applied it can produce *compression artifacts:* noticeable distortions in the media (Figure 4). For example, compressing an image too much can produce blurry and blocky shapes over the subject matter; compressing audio can change the sound quality and waveform of the music. Multiple participants describe scenarios where their model had artifacts, although they did not borrow this language from other types of media:

> *"Compressing like 90% can make things unstable with strange side effects. It's hard to figure out when you compress too much."* — E13

It is tempting to think of accuracy degradation as the only ML artifact; however, there are more subtle, even sinister implications depending on what a model is used for. For example, where some subpopulation of data is underrepresented in a dataset, e.g., at the tail of a distribution, it could be that ML compression techniques remove this tail, amplifying existing biases. Examples of this have been observed empirically in the few publications investigating robustness and evaluation of compressed models [31, 44, 45, 63, 77].

One specific compression artifact example story was told from multiple participants. In the case of an object detection model that needed to run at a high frame rate, during development teams noticed one day that the bounding boxes were jittering in their demo. This finding was surprising, since the metrics from their most recent model iteration were reporting no changes, but *"there were some weird side-effects"* [T5]. It was not until someone debugging the problem realized that they had applied quantization throughout the neural network, including the final prediction layer. This coarsening of the data at the output produced correct bounding boxes, but resulted in a poor user experience. Another participant had seen this before, and remarked if the output of a model is *"'fine grain,' don't quantize"* [T1].
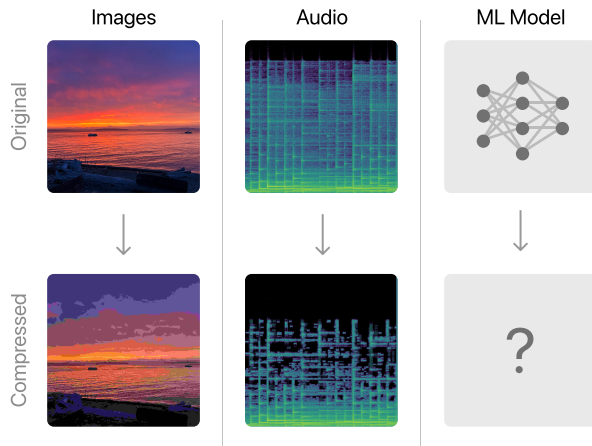
Figure 4: Illustrative examples of compression artifacts in images and audio, but what do machine learning model artifacts look like and how do we identify them?

Generalizing from this example, one learning that teams have found is that when a model's prediction needs to be continuous, smooth, or is user-perceivable at a high frame rate, it is best not to compress the final output layer [22]. Participants told us that other modeling tasks can be harder to optimize than classification [E5]. *"Regression models have been an absolute nightmare to compress"* [E7]. Anytime the output needs to be a continuous or floating point value, compression techniques can produce artifacts [E12].

5.6.5 Demoing User Experiences. ML-powered experiences can be dependent upon multiple models working in concert, and the *"best way to test compound [model] effects is having end-to-end evaluation"* [E10]. Another strategy to catch artifacts is testing models "as close to the metal as possible," i.e., loading the models on-device and building demo applications to run outside of lab environments. *"Since different hardware are not bit-wise accurate, metrics won't capture these changes"* [E12]. Another practitioner told us *"user experience differences are the really important cases to isolate"* [P5]. To find these user experience changes, teams have prioritized building demo applications where different models, e.g., a baseline model and a compressed model, can be dynamically toggled back and forth for testing [T4]. Some of these demos toggle between a model running on-device and a model running on a server [E12]. Interactive and live demos like these allow for ML teams to get feedback from other product stakeholders [E10]. They are particularly helpful for designers to get an overall "feel" of a model: interacting with a model in its intended environment to understand its capabilities and limitations [T4]:

> *"Get [the user experience] in the hands of people as fast as possible."* — T4

This notion of the "feel" of a model was described multiple times. If an *"ML engineer retrains [the model] and says it's better, we still need see if it feels better,"* [T4]. To try and attribute the feel of a model to actionable development steps, practitioners show debugging modes on-device to observe live, real-time charts of a model's

prediction and confidence, to help find edge cases and drill into specific errors that metrics may or may not capture [P11, P6, P7].

Strategy #7: Despite all the effort to create criteria and metrics to quantitatively measure and benchmark your model, your evaluation pipeline may not capture every aspect of your model's performance and behavior. Even if your model passes evaluation, move it on-device, in the intended environment in which it will run, and get it in the hands of users to demo. There is no other way to capture the feel of an ML-powered user experience.

---

6 DESIGN OPPORTUNITIES FOR EFFICIENT ML TOOLS AND INTERFACES

Given the unique challenges to efficient on-device ML, where can human-centered ML researchers, practitioners, and designers start to engage? As efficient machine learning techniques are driven forward by advances in hardware engineering and ML research, there remains a major barrier in practically applying these techniques for real-world features and user experiences.

Tools influence what is possible: *"there is a gap between what is possible with machine learning and what [tools] are being used,"* [E4]. Currently, the tooling for efficient on-device ML is underdeveloped, *"homegrown and ad-hoc,"* [E9]. Moreover, current tools focus on individual algorithms (Section 2.3.3) instead of the holistic process: *"this is a newer area, many tools are specific to a project and tend to be prototypes to prove feasibility,"* [E9]. As we have demonstrated in this paper, there is considerable design planning, experimentation, and strategy that experts use to make on-device ML a reality. Proper tools could help educate practitioners to develop these skills:

> *"Tooling is education products disguised as software. Better tools make it easy to do correct things and harder to do incorrect things."* — T2

We next share interdisciplinary HCI + ML research directions for supporting practitioners. While the space of tooling opportunities is wide and will evolve over time, we provide a few actionable recommendations for future tools and interfaces that are ready for development today.

6.1 Developing Intuition for Compression

Although modern libraries for machine learning make it easier than ever develop models, that does not mean practitioners know how to best train, evaluate, and deploy models. Machine learning is an inherently iterative and and empirical practice [7, 80], and developing intuition for how models learn and behave is a major competitive advantage over blindly tweaking hyper-parameters. Interactive playgrounds that provide a safe environment where practitioners can quickly build and test their ideas could be a great onboarding experience for learning about efficient ML and model compression. There is already precedent for this within machine learning, where interactive and educational tools help learners develop intuition around how certain models train and make predictions. Examples include include TensorFlow Playground [92] for small neural networks, CNN Explainer [109] for convolutional neural networks,

GANLab [52] for generative adversarial models, and Diffusion Explainer [57] for stable diffusion models. These interactives typically run fully in-browser, enabling learners to access and experiment with ML techniques without installing software to needing access to extra compute. Moreover, they have been incorporated into ML curricula to help people gain complementary, hands-on experience for specifics model categories.

Imagine a playground for efficient ML, where practitioners could learn and build intuition about different compression techniques [E12], model architectures [E7], and their effect on hardware [P9]. Perhaps this playground could be web-based, where a small model is loaded in the browser running live inference. Users could select different compression techniques, with varying degrees of strength, and see the impact on the model, its metrics, and its predictions. This could help practitioners define realistic budgets (Section 5.4), test different combinations of architectures and compression techniques (Section 5.5.1), and inspect the difference between compute-bound or memory-bound models (Section 5.5.4).

6.2 Comparing Across Compression Schemes

A common scenario in efficient model development is considering the trade-off between accuracy and performance (Section 5.6.1). While conventional ML development requires model comparison between architectures and hyper-parameters, the optimization metrics that practitioners must also navigate add additional complexity when moving models on-device [T5]. Consider the scenario of having a well-performing model that does not hit a size budget. What do you do next? You can try every compression technique possible, but how do you ultimately select the best model that balances between the desired trade-offs? In a similar process discussed in Section 5.6.3, one practitioner mentioned that they *"like to see an overview of a model's robustness before and after compression,"* [P8].

Better tooling to support model comparison could help practitioners compare a feasibility or baseline model (Section 5.4.1) against different compressed models to select the best model possible. There is rich opportunity for future work to investigate what to visualize and how. For example, flexible interfaces should handle visualizing the similarities and differences between models, their internals, their outputs, and applied compression techniques, where each technique not only has a suite of hyperparameters but can be also be combined with one another.

This line of work could also draw from existing work in experiment tracking. P10 described a project where the models they were developing were small, such that they could generate thousands of candidate models. This participant always compared candidates against an uncompressed baseline model, and maintained documentation tracking their experimental history, with a few notes per model, so that other stakeholders could make an informed model selection. However, this required diligent effort by the developer— experimental histories tend to exist across multiple documents and can result in long tables of values where patterns can be hard to discover. Future opportunities for tooling point to model tracking systems that can generate interactive reports to help practitioners observe their model development history, while helping them select the best model (from potentially thousands of models) that passes their accuracy threshold and maximizes performance.

6.3 Finding Model Bottlenecks for Targeted Compression

In Section 5.5.4 and Section 5.6.4, practitioners discuss analyzing a model layer-by-layer. *Targeted compression* is the practice of selectively optimizing specific components of a model, for example, individual layers in a neural network. Currently, practitioners described the process as tedious, manual, and time-consuming, where *"you must go layer by layer, operation by operation,"* [P11, P6, P7]. There exists a space of interfaces that could help practitioners look inside their models, by mapping metrics to specific layers of a network [T5], finding the performance bottlenecks in a network [E5], comparing the input and output of these bottleneck layers [E9], and selectively optimizing them.

Targeted compression tooling may also ease the difficulty of thinking at the hardware level. E6 gave an example: once an on-device model written in Python is compiled onto specialized hardware, the operations are expressed by what the hardware can support, and is likely not as readable as documented Python code. To complicate matters, low-level hardware operations may not have a one-to-one mapping back to the original Python code, due to optimizations in the compilation process. Better tools could help practitioners perform analytics on their models to find bottlenecks and easily traverse between different layers of abstraction (e.g., model code and hardware operations).

6.4 Evaluating Multi-model ML Systems

It is often assumed that model evaluation is done on a single model, isolated in a "lab environment." In practice, this is not the case. Not only are models integrated into larger apps or codebases, many modern ML-powered experiences are the result of multiple models working together. For example, models could be arranged in chains, where the outputs of one model are the inputs of another [E10]. If you compress one model, how does that impact downstream models? Recalling the concept of compounding error discussed in Section 5.6.2, small errors introduced in earlier models can compound to produce bigger errors by the end of a model chain [E10]. E3 emphasized this challenge, saying that the future of ML driven user experiences will be accomplished through multi-model systems. When multiple models are running simultaneously, each can be evaluated individually, but also must be considered as a whole, which makes it *"super hard to reason about,"* [E3]. Future tools that generalize and can evaluate multi-model machine learning systems will have a big impact in helping practitioners build, debug, and make sense of large data-driven applications.

6.5 Simplified Hardware Testing

One recommendation repeated by practitioners in this study was to measure model performance on device (Section 5.5.2). T4 emphasized that the performance of one model will like differ across different hardware (Section 5.5.3). However, testing on real hardware can be a major barrier. Due the variety of mobile devices and different versions of hardware, it can be hard to build for multiple hardware implementations simultaneously. The average ML developer (outside of a hardware company) may not have access to all versions of all devices their users might have. When it comes to future tooling opportunities, E6 says it best, *"the tools need to*

expose and take care of the cases for different hardware to maximize the hardware efficiency." While there is undoubtedly room for ML-hardware innovation here, for the purposes of this work assume this problem can be reduced to looping over a set of hardware and evaluating a model. With all these results, this problem could be cast as yet another comparison task, where tool designers need to enhance existing workflows to allow ML practitioners to track, organize, and see what hardware passed or failed certain criteria.

6.6 Automating (Some) Compression Experiments

While many of the practitioners we interviewed discussed creating efficient models as a human-in-the-loop iteration process, there are opportunities to automate applying different compression schemes and present results to developers. An apt analogy is what AutoML is to hyperparameter searching: instead of sequentially trying different compression schemes, perform a grid search and parallelize many different tests simultaneously. P3 was particularly excited about compression automation. Imagine mixed-initiative tools that take in a model and user-specified budgets, runs through a suite of compression techniques to find all possible experiments that satisfy the budgets, and finally presents a summary of recommended compression recipes for a practitioner to choose from. Perhaps through tools like these we could even learn something about the behavior of compression itself. However, we do not expect the process of creating efficient models to be completely automated. Throughout our study, experts made it clear that successful model compression is an iterative development process (Section 5.2), but leveraging the strengths of automation where appropriate suggests rich opportunity for mixed-initiative approaches for creating efficient models.

7 LIMITATIONS & VALIDITY

Optimizing and creating efficient models to run on mobile devices is still relatively new, and best practices are evolving alongside the rapid pace of ML research. Thus we expect not all the practices mentioned in this paper will stand the test of time. As discussed in Section 4, studying experts from a single organization naturally limits the perspective of this paper, therefore we also expect certain details of our findings may not generalize. Nonetheless, we believe that the high-level strategies for on-device ML shared in this work—including budget design, strategies for investing in model optimization, and proper compression evaluation—will generalize to be useful for many different kinds of devices, domains, and ML user experiences. We are eager to watch how this interdisciplinary area of research progresses in the future and reflect back on the practices outlined in this work to see which have remained, which have been removed, and which have been reinvented.

8 CONCLUSION

In this research we illustrated some of the pragmatic design considerations that go into efficient on-device machine learning. While this paper is far less technical than related work from the ML literature, we feel the interdisciplinary bridge to UX and product design are critical to bringing on-device ML into more approachable and popular practice. Through the results of this study, we are able to spotlight the holistic, end-to-end considerations that experts in

model compression have developed from translating ML research into intelligent, on-device ML experiences.

REFERENCES

[1] 2018. What is the team data science process? *Microsoft* (2018). https://learn.microsoft.com/en-us/azure/architecture/data-science-process/overview
[2] 2019. Design for AI. *IBM* (2019). https://www.ibm.com/design/ai/
[3] 2019. Human interface guidelines: Machine learning. *Apple Human Interface Guidelines* (2019). https://developer.apple.com/design/human-interface-guidelines/technologies/machine-learning/introduction
[4] 2019. People + AI guidebook. *Google* (2019). https://pair.withgoogle.com/guidebook/
[5] 2023. Machine learning workflow. *Google* (2023). https://cloud.google.com/ai-platform/docs/ml-solutions-overview
[6] Keivan Alizadeh, Iman Mirzadeh, Dmitry Belenko, Karen Khatamifard, Minsik Cho, Carlo C Del Mundo, Mohammad Rastegari, and Mehrdad Farajtabar. 2023. Llm in a flash: Efficient large language model inference with limited memory. *arXiv preprint arXiv:2312.11514* (2023).
[7] 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
[8] Saleema Amershi, Dan Weld, Mihaela Vorvoreanu, Adam Fourney, Besmira Nushi, Penny Collisson, Jina Suh, Shamsi Iqbal, Paul N Bennett, Kori Inkpen, et al. 2019. Guidelines for human-AI interaction. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 1–13. https://doi.org/10.1145/3290605.3300233
[9] Apple. 2023. *Optimizing models - Core ML Tools overview.* https://coremltools.readme.io/docs
[10] Apple. 2023. *Personalizing a model with on-device updates.* https://developer.apple.com/documentation/coreml/model_personalization/personalizing_a_model_with_on-device_updates
[11] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks* 5, 2 (1994), 157–166.
[12] Umang Bhatt, Alice Xiang, Shubham Sharma, Adrian Weller, Ankur Taly, Yunhan Jia, Joydeep Ghosh, Ruchir Puri, José MF Moura, and Peter Eckersley. 2020. Explainable machine learning in deployment. In *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency.* 648–657. https://doi.org/10.1145/3351095.3375624
[13] Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. 2021. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258* (2021).
[14] 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.
[15] Jagmohan Chauhan, Jathushan Rajasegaran, Suranga Seneviratne, Archan Misra, Aruna Seneviratne, and Youngki Lee. 2018. Performance characterization of deep learning models for breathing-based authentication on resource-constrained devices. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 2, 4 (2018), 1–24.
[16] Jiasi Chen and Xukan Ran. 2019. Deep learning with edge computing: A review. *Proc. IEEE* 107, 8 (2019), 1655–1674. https://doi.org/10.1109/jproc.2019.2921977
[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] 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

[20] Ruixuan Dai, Chenyang Lu, Michael Avidan, and Thomas Kannampallil. 2021. Respwatch: Robust measurement of respiratory rate on smartwatches with photoplethysmography. In *Proceedings of the International Conference on Internet-of-Things Design and Implementation*. 208–220.

[21] Fernando Delgado, Stephen Yang, Michael Madaio, and Qian Yang. 2021. Stake-holder participation in AI: Beyond "add diverse stakeholders and stir". *arXiv* (2021).

[22] 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

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

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

[25] 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

[26] Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, et al. 2022. Delta tuning: A comprehensive study of parameter efficient methods for pre-trained language models. *arXiv preprint arXiv:2203.06904* (2022).

[27] Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, et al. 2023. Parameter-efficient fine-tuning of large-scale pre-trained language models. *Nature Machine Intelligence* 5, 3 (2023), 220–235.

[28] Farah Fahim, Benjamin Hawks, Christian Herwig, James Hirschauer, Sergo Jin-dariani, 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

[29] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. 1996. The KDD process for extracting useful knowledge from volumes of data. *Commun. ACM* 39, 11 (1996), 27–34.

[30] Zihao Fu, Haoran Yang, Anthony Man-Cho So, Wai Lam, Lidong Bing, and Nigel Collier. 2023. On the effectiveness of parameter-efficient fine-tuning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 37. 12799–12807.

[31] Trevor Gale, Erich Elsen, and Sara Hooker. 2019. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574* (2019).

[32] 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

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

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

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

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

[37] 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

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

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

[40] Charles Hill, Rachel Bellamy, Thomas Erickson, and Margaret Burnett. 2016. Trials and tribulations of developers of intelligent systems: A field study. In *2016 IEEE Symposium on Visual Languages and Human-Centric Computing*. IEEE, 162–170. https://doi.org/10.1109/vlhcc.2016.7739680

[41] 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.

[42] Kenneth Holstein, Jennifer Wortman Vaughan, Hal Daumé III, Miro Dudik, and Hanna Wallach. 2019. Improving fairness in machine learning systems: What do industry practitioners need?. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 1–16. https://doi.org/10.1145/3290605.3300830

[43] Sungsoo Ray Hong, Jessica Hullman, and Enrico Bertini. 2020. Human factors in model interpretability: Industry practices, challenges, and needs. *Proceedings*

[44] *of the ACM on Human-Computer Interaction* 4, CSCW1 (2020), 1–26.

[44] Sara Hooker, Aaron Courville, Gregory Clark, Yann Dauphin, and Andrea Frome. 2019. What do compressed deep neural networks forget? *arXiv preprint arXiv:1911.05248* (2019).

[45] Sara Hooker, Nyalleng Moorosi, Gregory Clark, Samy Bengio, and Emily Denton. 2020. Characterising bias in compressed models. *arXiv* (2020). arXiv:2010.03058

[46] Aspen Hopkins and Serena Booth. 2021. Machine learning practices outside big tech: How resource constraints challenge responsible development. In *Proceed-ings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society*. 134–145.

[47] 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

[48] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. Lora: Low-rank adaptation of large language models. *International Conference on Learning Representations* (2022).

[49] Rui Hu, Yuanxiong Guo, Hongning Li, Qingqi Pei, and Yanmin Gong. 2020. Personalized federated learning with differential privacy. *IEEE Internet of Things Journal* 7, 10 (2020), 9530–9539.

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

[51] Linshan Jiang, Rui Tan, Xin Lou, and Guosheng Lin. 2021. On lightweight privacy-preserving collaborative learning for Internet of Things by independent random projections. *ACM Transactions on Internet of Things* 2, 2 (2021), 1–32.

[52] Minsuk Kahng, Nikhil Thorat, Duen Horng Polo Chau, Fernanda B Viégas, and Martin Wattenberg. 2018. Gan lab: Understanding complex deep generative mod-els using interactive visual experimentation. *IEEE Transactions on Visualization and Computer Graphics* 25, 1 (2018), 1–11. https://poloclub.github.io/ganlab/

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

[54] Jakub Konečný, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. 2016. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492* (2016).

[55] Aditya Kusupati, Vivek Ramanujan, Raghav Somani, Mitchell Wortsman, Prateek Jain, Sham Kakade, and Ali Farhadi. 2020. Soft threshold weight reparameteri-zation for learnable sparsity. In *International Conference on Machine Learning*. PMLR, 5544–5555.

[56] Steinar Kvale. 2006. Dominance through interviews and dialogues. *Qualitative Inquiry* 12, 3 (2006), 480–500.

[57] Seongmin Lee, Benjamin Hoover, Hendrik Strobelt, Zijie J Wang, ShengYun Peng, Austin Wright, Kevin Li, Haekyu Park, Haoyang Yang, and Duen Horng Chau. 2023. Diffusion explainer: Visual explanation for text-to-image stable diffusion. *arXiv preprint arXiv:2305.03509* (2023).

[58] Youpeng Li, Xuyu Wang, and Lingling An. 2023. Hierarchical clustering-based personalized federated learning for robust and fair human activity recogni-tion. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 7, 1 (2023), 1–38.

[59] Vladislav Lialin, Vijeta Deshpande, and Anna Rumshisky. 2023. Scaling down to scale up: A guide to parameter-efficient fine-tuning. *arXiv preprint arXiv:2303.15647* (2023).

[60] Q Vera Liao, Daniel Gruen, and Sarah Miller. 2020. Questioning the AI: Informing design practices for explainable AI user experiences. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 1–15. https://doi.org/10.1145/3313831.3376590

[61] Daniyal Liaqat, Mohamed Abdalla, Pegah Abed-Esfahani, Moshe Gabel, Ta-tiana Son, Robert Wu, Andrea Gershon, Frank Rudzicz, and Eyal De Lara. 2019. WearBreathing: Real world respiratory rate monitoring using smartwatches. *Pro-ceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 3, 2 (2019), 1–22.

[62] Edgar Liberis and Nicholas D Lane. 2023. Differentiable neural network pruning to enable smart applications on microcontrollers. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 6, 4 (2023), 1–19.

[63] Lucas Liebenwein, Cenk Baykal, Brandon Carter, David Gifford, and Daniela Rus. 2021. Lost in pruning: The effects of pruning neural networks beyond test accuracy. *Proceedings of Machine Learning and Systems* 3 (2021), 93–138.

[64] 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 Communica-tions Surveys & Tutorials* 22, 3 (2020), 2031–2063. https://doi.org/10.1109/comst.2020.2986024

[65] Sicong Liu, Bin Guo, Ke Ma, Zhiwen Yu, and Junzhao Du. 2021. AdaSpring: Context-adaptive and runtime-evolutionary deep model compression for mo-bile applications. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 5, 1 (2021), 1–22.

[66] Michael Madaio, Lisa Egede, Hariharan Subramonyam, Jennifer Wort-man Vaughan, and Hanna Wallach. 2022. Assessing the fairness of AI systems: AI practitioners' processes, challenges, and needs for support. *Proceedings of*

the ACM on Human-Computer Interaction 6, CSCW1 (2022), 1–26.

[67] Michael A Madaio, Luke Stark, Jennifer Wortman Vaughan, and Hanna Wallach. 2020. Co-designing checklists to understand organizational challenges and opportunities around fairness in AI. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 1–14.

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

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

[70] Rahul Mishra and Hari Prabhat Gupta. 2023. Designing and training of lightweight neural networks on edge devices using early halting in knowledge distillation. *IEEE Transactions on Mobile Computing* (2023).

[71] Rahul Mishra, Hari Prabhat Gupta, and Tanima Dutta. 2020. Teacher, trainee, and student based knowledge distillation technique for monitoring indoor activities. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*. 729–730.

[72] Ceena Modarres, Mark Ibrahim, Melissa Louie, and John Paisley. 2018. Towards explainable deep learning for credit lending: A case study. *arXiv* (2018).

[73] Steven Gonzalez Monserrate. 2022. The cloud is material: On the environmental impacts of computation and data storage. *MIT Case Studies in Social and Ethical Responsibilities of Computing* Winter 2022 (Jan 2022). https://doi.org/10.21428/2c646de5.031d4553

[74] Rajiv Movva, Sidhika Balachandar, Kenny Peng, Gabriel Agostini, Nikhil Garg, and Emma Pierson. 2023. Large language models shape and are shaped by society: A survey of arXiv publication patterns. *arXiv preprint arXiv:2307.10700* (2023).

[75] 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

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

[77] Kelechi Ogueji, Orevaoghene Ahia, Gbemileke Onilude, Sebastian Gehrmann, Sara Hooker, and Julia Kreutzer. 2022. Intriguing properties of compression on multilingual models. *arXiv preprint arXiv:2211.02738* (2022).

[78] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*. PMLR, 1310–1318.

[79] Samir Passi and Steven J Jackson. 2018. Trust in data science: Collaboration, translation, and accountability in corporate data science projects. *Proceedings of the ACM on Human-Computer Interaction* 2, CSCW (2018), 1–28.

[80] 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

[81] David Piorkowski, Soya Park, April Yi Wang, Dakuo Wang, Michael Muller, and Felix Portnoy. 2021. How ai developers overcome communication challenges in a multidisciplinary team: A case study. *Proceedings of the ACM on Human-Computer Interaction* 5, CSCW1 (2021), 1–25.

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

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

[84] PyTorch. 2019. *Sparsity*. https://pytorch.org/docs/stable/sparse.html

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

[86] Bogdana Rakova, Jingying Yang, Henriette Cramer, and Rumman Chowdhury. 2021. Where responsible AI meets reality: Practitioner perspectives on enablers for shifting organizational practices. *Proceedings of the ACM on Human-Computer Interaction* 5, CSCW1 (2021), 1–23.

[87] Samantha Robertson, Tonya Nguyen, and Niloufar Salehi. 2021. Modeling assumptions clash with the real world: Transparency, equity, and community challenges for student assignment algorithms. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–14.

[88] Nithya Sambasivan, Shivani Kapania, Hannah Highfill, Diana Akrong, Praveen Paritosh, and Lora M Aroyo. 2021. "Everyone wants to do the model work, not the data work": Data cascades in high-stakes AI. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–15. https://doi.org/10.1145/3411764.3445518

[89] 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

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

[91] 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.

[92] Daniel Smilkov, Shan Carter, D Sculley, Fernanda B Viegas, and Martin Wattenberg. 2016. Direct-manipulation visualization of deep networks. In *ICML*

[93] Irene Solaiman, Zeerak Talat, William Agnew, Lama Ahmad, Dylan Baker, Su Lin Blodgett, Hal Daumé III, Jesse Dodge, Ellie Evans, Sara Hooker, et al. 2023. Evaluating the social impact of generative AI systems in systems and cociety. *arXiv preprint arXiv:2306.05949* (2023).

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

[95] 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

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

[97] 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

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

[99] Nenad Tomašev, Julien Cornebise, Frank Hutter, Shakir Mohamed, Angela Picciariello, Bec Connelly, Danielle CM Belgrave, Daphne Ezer, Fanny Cachat van der Haert, Frank Mugisha, et al. 2020. AI for social good: Unlocking the opportunity for positive impact. *Nature Communications* 11, 1 (2020), 2468.

[100] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. 2021. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*. PMLR, 10347–10357.

[101] Marcos Treviso, Ji-Ung Lee, Tianchu Ji, Betty van Aken, Qingqing Cao, Manuel R Ciosici, Michael Hassid, Kenneth Heafield, Sara Hooker, Colin Raffel, et al. 2023. Efficient methods for natural language processing: A survey. *Transactions of the Association for Computational Linguistics* 11 (2023), 826–860.

[102] 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).

[103] Pavan Kumar Anasosalu Vasu, James Gabriel, Jeff Zhu, Oncel Tuzel, and Anurag Ranjan. 2023. An improved one millisecond mobile backbone. (2023).

[104] 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]

[105] Dakuo Wang, Josh Andres, Justin D Weisz, Erick Oduor, and Casey Dugan. 2021. Autods: Towards human-centered automation of data science. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–12.

[106] Dakuo Wang, Q Vera Liao, Yunfeng Zhang, Udayan Khurana, Horst Samulowitz, Soya Park, Michael Muller, and Lisa Amini. 2021. How much automation does a data scientist want? *arXiv* (2021).

[107] Dakuo Wang, Justin D Weisz, Michael Muller, Parikshit Ram, Werner Geyer, Casey Dugan, Yla Tausczik, Horst Samulowitz, and Alexander Gray. 2019. Human-AI collaboration in data science: Exploring data scientists' perceptions of automated AI. *Proceedings of the ACM on Human-Computer Interaction* 3, CSCW (2019), 1–24.

[108] Yanfei Wang, Zhiwen Yu, Sicong Liu, Zimu Zhou, and Bin Guo. 2023. Genie in the model: Automatic generation of human-in-the-loop deep neural networks for mobile applications. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 7, 1 (2023), 1–29.

[109] Zijie J. Wang, Robert Turko, Omar Shaikh, Haekyu Park, Nilaksh Das, Fred Hohman, Minsuk Kahng, and Duen Horng (Polo) Chau. 2021. CNN explainer: Learning convolutional neural networks with interactive visualization. In *IEEE Transactions on Visualization and Computer Graphics*. IEEE. https://doi.org/10.1109/TVCG.2020.3030418

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

[111] 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

[112] Maximiliane Windl, Sebastian S Feger, Lara Zijlstra, Albrecht Schmidt, and Pawel W Wozniak. 2022. 'It is not always discovery time': Four pragmatic approaches in designing AI systems. In *CHI Conference on Human Factors in Computing Systems*. 1–12.

[113] Rüdiger Wirth and Jochen Hipp. 2000. CRISP-DM: Towards a standard process model for data mining. In *Proceedings of the 4th international conference on the practical applications of knowledge discovery and data mining*, Vol. 1. Manchester, 29–39.

[114] Austin P. Wright, Zijie J. Wang, Haekyu Park, Grace Guo, Fabian Sperrle, Mennatallah El-Assady, Alex Endert, Daniel Keim, and Duen Horng Chau. 2020. A comparative analysis of industry human-AI interaction guidelines. *Workshop on Trust and Expertise in Visual Analytics at IEEE VIS* (2020).

Workshop on Vis for Deep Learning.

[115] 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.

[116] Canwen Xu and Julian McAuley. 2023. A survey on model compression and acceleration for pretrained language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 37. 10566–10575.

[117] Xuhai Xu, Jun Gong, Carolina Brum, Lilian Liang, Bongsoo Suh, Shivam Kumar Gupta, Yash Agarwal, Laurence Lindsey, Runchang Kang, Behrooz Shahsavari, et al. 2022. Enabling hand gesture customization on wrist-worn devices. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. 1–19.

[118] Qian Yang. 2018. Machine learning as a UX design material: How can we imagine beyond automation, recommenders, and reminders?. In *AAAI Spring Symposium Series*.

[119] Qian Yang, Alex Scuito, John Zimmerman, Jodi Forlizzi, and Aaron Steinfeld. 2018. Investigating how experienced UX designers effectively work with machine learning. In *Proceedings of the 2018 Designing Interactive Systems Conference*. 585–596.

[120] Dixi Yao, Liyao Xiang, Zifan Wang, Jiayu Xu, Chao Li, and Xinbing Wang. 2021. Context-aware compilation of dnn training pipelines across edge and cloud. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 5, 4 (2021), 1–27.

[121] Nur Yildirim, Alex Kass, Teresa Tung, Connor Upton, Donnacha Costello, Robert Giusti, Sinem Lacin, Sara Lovic, James M O'Neill, Rudi O'Reilly Meehan, et al. 2022. How experienced designers of enterprise applications engage AI as a design material. In *CHI Conference on Human Factors in Computing Systems*. 1–13.

[122] 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

[123] Sabah Zdanowska and Alex S Taylor. 2022. A study of UX practitioners roles in designing real-world, enterprise ML systems. In *CHI Conference on Human Factors in Computing Systems*. 1–15.

[124] 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.

[125] 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

[126] 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

[127] 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).

[128] Yong Zhao, Jinyu Li, and Yifan Gong. 2016. Low-rank plus diagonal adaptation for deep neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 5005–5009.

[129] Peining Zhen, Hai-Bao Chen, Yuan Cheng, Zhigang Ji, Bin Liu, and Hao Yu. 2021. Fast video facial expression recognition by a deeply tensor-compressed LSTM neural network for mobile devices. *ACM Transactions on Internet of Things* 2, 4 (2021), 1–26.

[130] Ce Zhou, Qian Li, Chen Li, Jun Yu, Yixin Liu, Guangjing Wang, Kai Zhang, Cheng Ji, Qiben Yan, Lifang He, et al. 2023. A comprehensive survey on pretrained foundation models: A history from BERT to ChatGPT. *arXiv preprint arXiv:2302.09419* (2023).

[131] 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

[132] 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

A INTERVIEW QUESTIONS

The list of questions prepared for each interview participant.

Background ML Information

   Q1. What is your team?

   Q2. What is your role?

   Q3. How many years of ML experience do you have?

   Q4. How many years of efficient ML experience do you have?

General Compression Questions

   Q5. Describe the overall use case for model compression in your work. What are your main motivations to use model compression? Why do you care about model compression?

   Q6. Model compression details:

      Q6.1 Which model compression techniques do you use, and why? Are there trade offs or preferences?

      Q6.2 Do you do compression as a final step or as a part of the training process?

      Q6.3 In your experience, are there any pitfalls people applying compression should be aware of?

      Q6.4 How did you figure out your budgets and how did you satisfy them?

   Q7. Do you compare compressed models against baseline models? If so, how, and what do you look for, e.g., power and performance, other metrics, or user experience changes?

Compression Tooling Questions

   Q8. What tools or visualizations do you use in your compression work? Please be specific, e.g., specific charts, views, or metrics. How do you use these tools?

   Q9. What do you like about these tools?

   Q10. What features or future tools would help you conduct better model compression work?