Transformers on Consumer Hardware: A Critical Perspective of TinyML Optimization Techniques and Open Problems

Anonymous authors
Paper under double-blind review

Abstract

Deep learning models that are based on the transformer architecture have a reputation for requiring large compute resources for training and inference. This requirement has placed transformer-based models, such as large language and other generative AI models, beyond the reach of low-resource devices which make up most of the computer systems in the world. Conversely, machine learning is currently experiencing a revolution of a smaller sort, in which techniques under the umbrella of "TinyML" are optimizing feed-forward and convolutional models to run successfully on these low-resource devices. Gated access to high-performance compute clusters, rising compute costs, and lack of general access have driven research in combining these two fields. Today, TinyML techniques such as pruning, quantization, and software-hardware co-design are being applied to transformer-based models to deploy transformer-based models to low-resource and edge devices. Analysis of the surveyed works reveals that the techniques applied are largely orthogonal to one another, that knowledge distillation is significantly underrepresented, and that edge training remains rare. The most accessible path toward progress lies in combining the independently developed contributions already present in the literature.

1 Introduction

Deep learning models have achieved incredible results in many areas, including vision (Krizhevsky et al., 2012), natural language processing (Vaswani et al., 2017), and generative tasks (Rombach et al., 2022). These achievements continue to move beyond human-level performance in various tasks and further into the realm of the superhuman. Generative AI tools are being increasingly used for professional services since the release of GPT-3 and show no sign of slowing down; instead, the pace is increasing (Warren et al., 2025). As the complexity of deep learning models has grown, the amount of energy consumed in the training and utilization of these models has increased with it. In 2022, 2% of global energy consumption was attributed to AI, and this was on pace to increase to 10% by 2025. Indeed, a 2024 projection of electricity demand estimated that total AI energy consumption in 2026 would be equal to the entire energy consumption of Sweden or Germany (Kshetri, 2024). As a result, considerable effort is being applied to reduce energy consumption through model optimization that can be accomplished by model compression, combinations of compression techniques and novel hardware acceleration, and deployment strategies for model inference.

Systems used to train and deploy state-of-the-art generative models have moved beyond the resources available to the average consumer, moving into large data centers and compute clusters and away from the local desktop environment. Foundational generative models possess hundreds of billions of parameters and have massive VRAM requirements making them impractical for local training (Scao et al., 2022). Compression techniques are essential for both the training of these models and efficient inference on consumer devices with a fraction of the available memory space. While resource requirements for deep models continue to rise, the production of low-resource devices is also rising. These devices, often deployed for edge computation, make use of local sensors to process the world around them, can run for long periods of time with minimal power requirements without needing to be recharged (if not connected to a local power source such as solar energy), and gain a unique perspective by having a direct view of the world. However, many deep learning models have resource requirements that are too large for inference tasks on embedded devices, much less

training. This creates a gap in the ability to utilize these deep learning models on the many edge devices that now exist. These devices have a fraction of available memory even to that of desktop hardware.

Tiny Machine Learning (TinyML) has gained traction in recent years. TinyML is a paradigm that facilitates running deep learning models on edge devices with minimal processor and memory resources. It is driven by the need to expand the intelligence capabilities of deep learning into a wide range of applications that were not viable due to the high power and computation requirements of standard deep learning models. This has produced many examples of deep learning models, particularly convolutional neural networks (CNNs), that can be deployed and even trained on edge devices (Lin et al., 2022a). Edge devices can offer key advantages compared to cloud deployment including reduced latency, offline capability, privacy and security concerns, low-energy consumption, and cost reduction (Abadade et al., 2023).

Models developed in the TinyML space have shown enormous promise in developing and deploying deep learning models at the edge (Capogrosso et al., 2024), but many recent advances in generative AI utilize approaches such as diffusion (Rombach et al., 2022), GANs (Goodfellow et al., 2014), and the transformer (Vaswani et al., 2017) which are not necessarily feasible for tiny devices. Since its introduction in 2017, the transformer has become a fundamental baseline for many types of generative models, and it lies at the heart of the most common tools for AI assistants (Devlin et al., 2019). With more efficient training leading to state-of-the-art results, models have tended to grow larger, both in parameter count and in the data processed during training.

Transformers form the heart of large language models (LLMs), and their ability to scale with hardware has allowed these models to be trained on massive datasets to form a foundational basis for language (Sengar et al., 2025). The foundational models underpinning most LLMs require incredible resource costs to train (JarvisLabs, 2026; Scao et al., 2022). Inference costs, while not as expensive as training, often require powerful desktop computers to generate results in any meaningful span of time. This poses a challenge for the deployment and use of these types of models on low-resource devices, and as a result, limits the ability to utilize the transformer as the core approach for tiny devices.

1.1 Research Questions Motivating the Survey

From the above, we can pose several questions:

RQ1. How are transformers being used in edge and low-resource computing?

RQ2. What TinyML techniques are being applied to transformer architectures for edge training and deployment?

RQ3. What limitations remain in current TinyML transformer research?

1.2 Outline of the Survey

To investigate these questions, this paper presents a *narrative survey* of TinyML techniques applied to transformer-based models for edge training and inference. Section 2 touches on the transformer architecture and an overview of TinyML. Section 3 explores this survey's methodology and how papers were organized. Sections 4 through 6 examine several compression techniques commonly used in TinyML being used in current works. Section 7 presents cross-paper analysis that examines trends, limitations, and research gaps in the cited works. Finally, Section 8 ends the paper with a look at future research directions we can take and overall conclusions.

2 Background

2.1 The Transformer Architecture

The transformer is a neural network architecture that has emerged from the need to process sequential data efficiently. This originated in the field of natural language processing (NLP) where it is necessary to be able

to process both the words (tokens) and the context in which they exist. Early attempts using artificial neural networks involved recurrent neural networks (RNNs) and in particular long short-term memory networks (LSTMs) which would process the different tokens in order. Other methods use attention, a method which maps the influence that each token in a sequence has upon another token based on the the relevance/weight between each other (Bahdanau et al., 2014). Vaswani et al. (2017) introduced multi-headed attention (MHA), a key component of the transformer, as a way to learn multiple representations of the tokens that can exist in parallel. This architecture is represented in Figure 1.

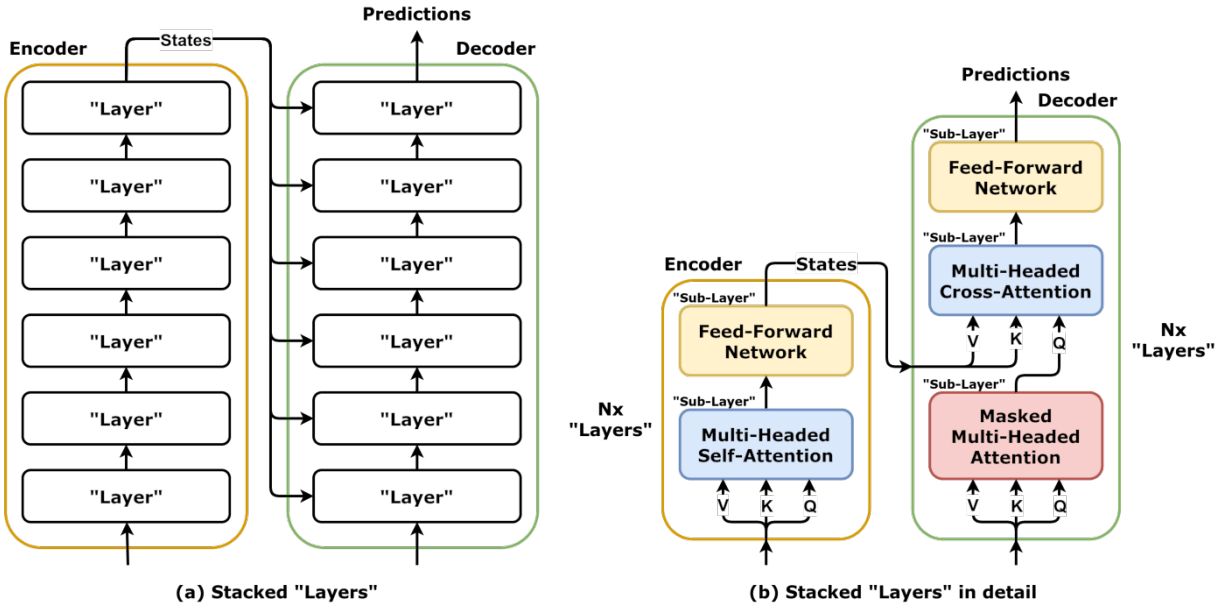

Figure 1: A view of the original transformer presented by (Vaswani et al., 2017). This shows both the multi-headed attention layout and the encoder-decoder layout.

This approach is highly adaptable and scalable. The initial approach utilized an encoder-decoder approach to encode text in one language and decode it in a different language. Later updates to this concept, such as in the work of Devlin et al. (2019), move away from an encoder-decoder design to create encoder-only setups with the aim of better capturing context within language.

Training for NLP tasks often requires massive corpora of data in order to capture the often subtle context of one token's relationship to another. The predictably scalable nature of the transformer model lends itself well to this requirement with state-of-the-art models being trained on tremendous resources (Scao et al., 2022; Kaplan et al., 2020). When training a large language model in this way, the process is usually split into two steps, pretraining and fine-tuning. Pretraining involves allowing the model to learn a broad sense of language as it is trained on trillions of token examples. Then, a fine-tuning phase is applied where the model is further trained on a smaller, more specialized dataset. The performance of BERT has allowed it to become a centerpiece in NLP (Koroteev, 2021).

Inference on this type of model would ideally run in parallel as well, dividing the labor to dedicated resources. The substantial resource requirements required can hinder or prevent inference on low-resource, edge devices, which possess a tiny fraction of the compute that compute clusters have and often lack dedicated parallel processing support, preventing the efficient performance on these devices. Thus, efforts to reduce either the parameter count or the number of activations are critical in improving performance on inference tasks (Yang et al., 2023a).

2.2 TinyML Overview

Tiny Machine Learning emerged in 2018 as a paradigm that allows for machine learning on edge devices with minimal processor and memory requirements. In this field, researchers specifically target devices that only consume a few milliwatts or less. Capogrosso et al. (2024) divides TinyML into three workflows: **ML-oriented**, **HW-oriented**, and **co-design**. In ML-oriented workflows, suitable models are designed and implemented to optimize for a particular problem domain for deployment on low-resource hardware. This workflow focuses on model design and compression techniques to introduce sparsity and reduce resource requirements such as pruning, quantization, knowledge distillation, and neural architecture search. In HW-oriented workflows, non-prearranged hardware is utilized to run deep learning models. This workflow focuses on novel hardware designs to accelerate ML inference. Finally, co-design workflows aim to unite these two with both custom, optimal model designs and unique hardware solutions. While the goal of TinyML is to deploy ML models at the edge, it also makes sense to examine hardware architectures that are not commonly associated with TinyML, although not at the level of compute commonly associated with AI. This type of relationship can be seen in Figure 2. Moore's Law of increasing performance/decreasing cost of hardware components allows us to examine these models on consumer-oriented hardware which is compute-limited and warrants examination.

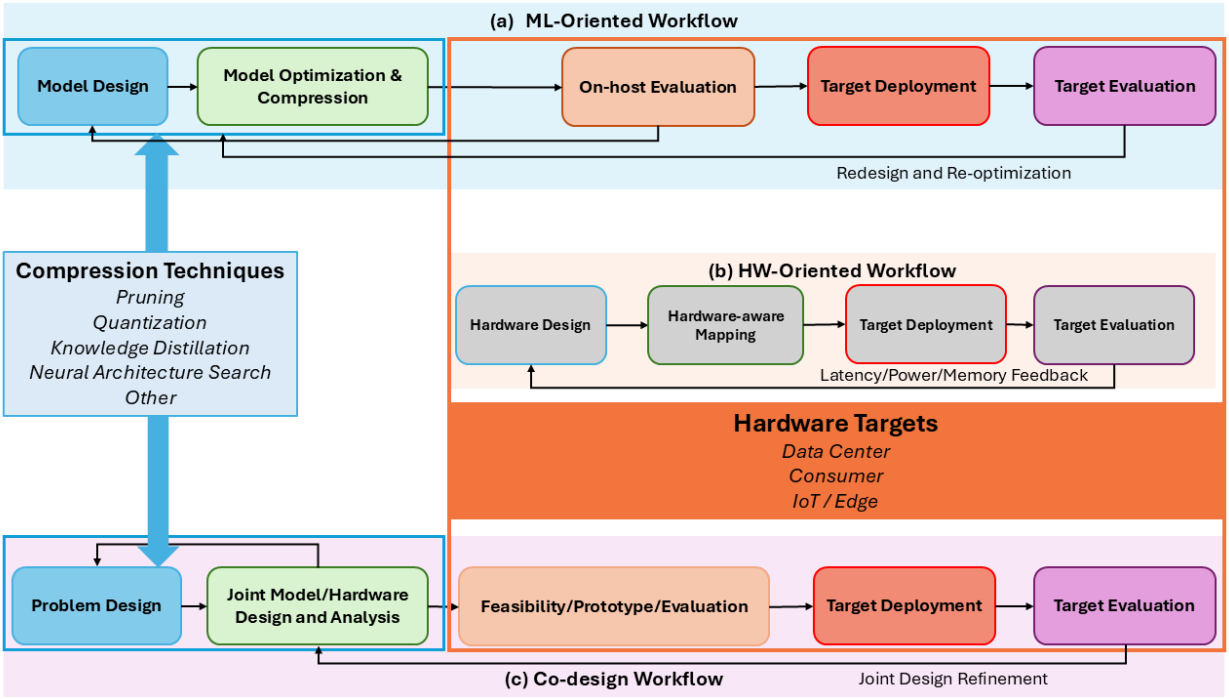

Figure 2: TinyML optimization techniques and hardware platforms examined in this survey based on their TinyML workflows. Each workflow (a/b/c) captures a process for ML design and deployment.

Currently, CNNs are often used as the model architecture for TinyML. As edge devices are often deployed to capture visual data, this association makes sense. However, as transformers expand in scope beyond NLP and into computer vision tasks (Dosovitskiy et al., 2020), applying state-of-the-art transformer models to edge devices makes sense. The drastically reduced performance of edge devices makes this a difficult task, but research in the last few years is bringing transformer architectures, TinyML, and edge devices together in both training and inference tasks. Moreover, while the strict definition of TinyML limits the resources available to a few kilobytes, the concept of an edge device has expanded. Edge devices can range from tiny microcontrollers that serve as access points into a larger network, but could also refer to local hardware with stronger hardware including dedicated graphics.

Expanding beyond the strict definition of TinyML allows for research that can operate more within the consumer environment. This would focus on devices within the acquisition range of the average consumer. This is a broad range, from the low-end Raspberry Pi system-on-a-chip (SBC) to consumer-grade laptops, but which we label as *low-resource computing* with the additional restriction of being able to be taken and operated on offline. It is this range that we find interesting and we discuss it in this survey.

3 Survey Methodology

This survey is organized to examine the intersection of TinyML techniques with with the use of the transformer architecture. Other surveys we have investigated, such as Saha & Xu (2025), focus specifically on the vision transformer (ViT) at the edge (Dosovitskiy et al., 2020). Capogrosso et al. (2024) showed the traditional workflow-oriented order for TinyML: ML-oriented, HW-oriented, and co-design workflows, which offer different approaches for implementing TinyML with available resources. The aim of this survey is to examine approaches that preserve offline inference in transformer implementations on low-resource hardware. This differs from other surveys in that we use a broader definition of low-resource than they do. Many generative models in use today rely on a service-based interface to counter the high inference resource requirements and make the generative models more accessible to the public. TinyML often takes this offline approach as its implementers process data on-device. We identify structural gaps in approaches that focus on consumer-available hardware that many users already possess. For a more exhaustive mapping of techniques to different MCU hardware, refer to Saha & Xu (2025).

To investigate research that has taken this approach we consulted Google Scholar to search for recent advances in the past five years. Taking cues from the specific techniques highlighted in Capogrosso et al. (2024), we scanned this database for papers that highlight work done intersecting transformer models with TinyML techniques using specific terms. The results of these searches, along with their terms, are presented in Table 1.

Search Term	Raw Results	Filtered Results
tinyml transformers +pruning -survey -review	159	2
tinyml transformers +quantization -survey -review	223	1
tinyml transformers +knowledge distillation -survey -review	168	1
tinyml transformers +neural architecture search -survey -review	221	1
Total Papers:	755	5

Table 1: Google Scholar Search Parameters for Various TinyML Techniques

First, we begin with a raw search through the database pulling the set of results returned with each query in Table 1. Then, we remove results that do not show progress in each area. As such, we generally filter for the following criteria: papers with citation counts greater than 20 citations, papers in the English language, papers that are published in a conference or journal, and papers publishing non-survey novel research between 2020 and 2025. This filtering process is illustrated in Figure 3. This filtered list of papers was then reviewed for content, but we found that some results were misclassified or improperly classified, some results still contained survey or benchmark papers and were removed, and some papers utilized multiple techniques and the duplicate entries were removed. Finally, many of the papers that the database returned as a "transformer" paper only made reference to the transformer in related works, but their contribution targeted a different architecture.

As a result, only five papers were found to be valid for the criteria set out in this survey. However, upon review of the papers that fit the criteria and a review of papers that did not pertain to transformers or to local, offline devices, additional papers that are applicable to this research area were discovered and added to this survey for a total of 19 papers reviewed. These papers were selected through a rigorous manual search through Google Scholar from the citations of the five works filtered. These works were selected for their application of TinyML techniques on Transformer models. These papers are presented in Table 2.

Works reviewed in this survey will showcase models that perform inference tasks without external connections, and models in this survey are targeted for use by commercial hardware. This limits the scope of the

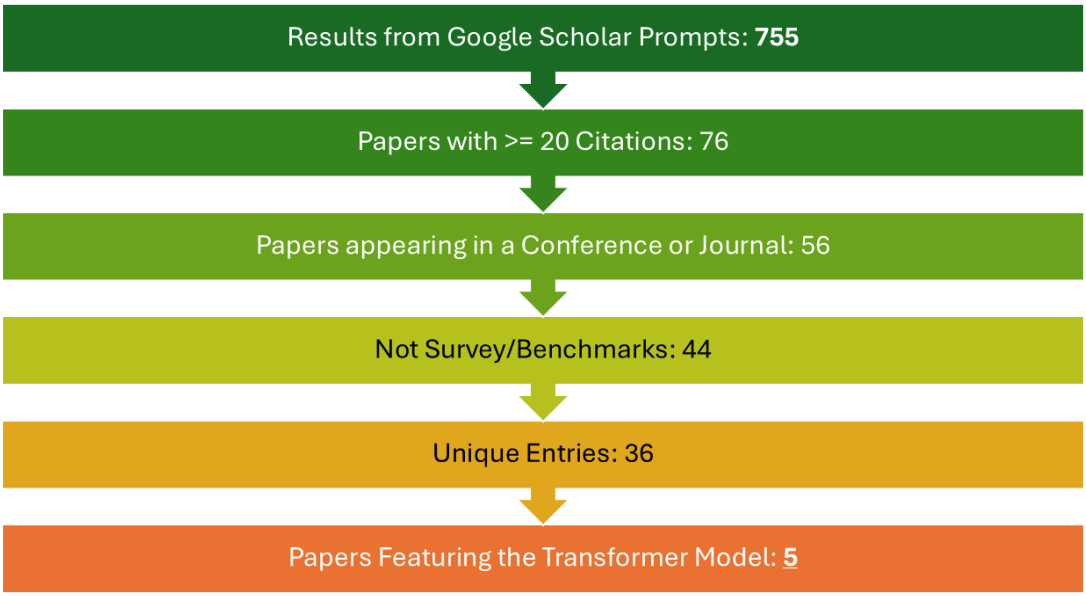

Figure 3: Filter Process for Paper Selection in this Survey

works cited here to hardware that the average user can order, including, but not limited to: commercial computers available to the public, low-resource specialized hardware available to the public (similar to the NVIDIA Jetson family of products), and edge devices such as the Raspberry Pi or the Arduino family of products. The overall goal of this survey is to examine the TinyML techniques used in models on local, consumer, hardware.

Organization As mentioned in Section 1.2, the remainder of this survey will examine different TinyML approaches. It will first approach the ML-oriented workflow by examining several compression techniques, neural architecture search techniques, and one work on chain-of-thought compression. Then it will review a selection of hardware/software co-design techniques that are being developed and examine both the targeted hardware and the compression techniques together. Each section is accompanied by analysis of the technique being used and which part of the model is being optimized. Finally, this survey will present novel conclusions of the common elements of the cited works and understanding possible future efforts that this research space could pursue. The works formally reviewed in this survey are located in Table 2.

4 ML-Oriented Approaches for Compression

Neural network models often suffer from significant redundancy in the number of parameters they have, and these parameters are often not necessary for performance (Denil et al., 2013). The structured nature of deep neural networks naturally leads to this over-design. It has been shown in Frankle & Carbin (2018) that a neural network contains sub-networks that, when trained in isolation, match the performance of the fully connected network. It was found that these sub-networks can be as small as 10-20% of the size of the original network. Furthermore, these networks have parameters and activations that are normally trained in 32-bit precision, but can often be reduced to 8- or 16-bit precision with minimal reduction in accuracy (Nagel et al., 2021). Finally, training smaller models from the logits of a larger, more complex model can discover these sub-networks as well (Hinton et al., 2015). Several TinyML techniques are well-suited to this task, and it is important to understand how each of these techniques on transformer-based models are used in the literature.

Citation	Techniques	Section
Ahia et al. (2021)	Pruning	4.1
Kwon et al. (2022)	Pruning	4.1
Sun et al. (2024)	Pruning	4.1
Sharifirad et al. (2025)	Pruning	4.1
Elliott et al. (2021)	Quantization	4.2
Lin et al. (2024)	Quantization	4.2
Busia et al. (2025)	Quantization	4.2
Gu et al. (2023)	Knowledge Distillation	4.3
So et al. (2019)	Neural Architecture Search	4.4
Jing et al. (2020)	Neural Architecture Search	4.4
Wang et al. (2020)	Neural Architecture Search	4.4
Busia et al. (2022)	Other	4.5
Zhu et al. (2023)	Other	4.5
Kang et al. (2025)	Other	4.5
Marchisio et al. (2023)	Co-Design	5
Yang et al. (2023b)	Co-Design	5
Song et al. (2025)	Co-Design	5
Han et al. (2024)	Deployment	6
Jung et al. (2025)	Deployment	6

Table 2: Reviewed Papers in this Survey with TinyML Technique

4.1 Pruning

Pruning is a compression technique where weights and/or activations of a neural network are removed from the network itself (Han et al., 2015), reducing the amount of processing required to travel the network. This has a downside of potentially reducing the accuracy of the model, and often the network must be retrained after unnecessary weights are removed. This is a massive challenge for transformer-based models, which often have parameter counts in the tens of billions for state-of-the-art models. Different works explore this problem and offer different solutions.

In Kwon et al. (2022), the authors introduce a three-stage structured pruning method that requires no pretraining and achieves a 60-70% reduction in FLOPs with only a 1% accuracy reduction. This approach prunes both attention heads in MHA and filters in the model's feed-forward layers. The authors formulate transformer pruning as a constrained optimization problem of a mask m with a FLOPs or latency constraint C that aims to mask out certain portions of the model for pruning as seen below.

$$\arg\min_{m} \mathcal{L}(m) \mid Cost(m) \leq C \tag{1}$$

The first process of this method is a mask-search that attempts to find attention heads to prune. For FLOPs optimization, this is done by first finding an approximate Fisher Information Matrix I based on a sample dataset. However, as finding the full matrix is intractable in transformer models, the authors make the assumption that I is diagonal. The authors then interpret the diagonal elements of I as the importance score of the head/filter and prune the diagonal elements with the lowest importance scores. The authors also approach the problem of latency constraints. For this approach minimization objective (1) is modified to approximate the latency cost as shown in (2), due to its non-linear nature with respect to the number of remaining heads or filters after pruning. The function LAT indicates the latency of an MHA/FFN layer after pruning.

$$\arg\min_{\mathrm{m}} \sum_{i \in Z(\mathrm{m})} \mathcal{I} \text{ s.t. } \sum_{l=1}^{L} \mathrm{LAT}(\mathrm{m}_l^{\mathrm{MHA}}) + \sum_{l=1}^{L} \mathrm{LAT}(\mathrm{m}_l^{\mathrm{FFN}}) \leq C, \tag{2}$$

The second process is a rearrangement of the Fisher matrix to ensure that the interactions of different mask variables are considered by partitioning the matrix into blocks. This is done by using the masks found in the first process as a starting point of a greedy search to determine how many heads/filters to prune in each layer. The final process is to tune the mask values, which are restricted to either 1 or 0 in the first two approaches, to any real values such that the pruned model does not lose accuracy.

The authors compare their pruning method against four others that achieve comparably good accuracy and all methods are analyzed for their retraining costs. The four comparisons range from 4 to 40 training epochs and between 5 and 33 hours of retraining time. This method does *not* require retraining, finishing in less than one minute. Each method, once retrained, achieves similar levels of accuracy, including the authors' method.

Sun et al. (2024) also establishes a non-retraining approach to pruning. This emerges from two key findings: (i) that magnitude pruning (Zhu & Gupta, 2017) fails dramatically on LLMs even with relatively low levels of sparsity, and (ii) that once LLMs reach a certain scale, a small set of hidden features emerges with significantly larger magnitudes than the remaining ones.

Starting from magnitude pruning, the authors take it one step further, crafting an alternative score ($\mathbf{S}_{i,j} = |\mathbf{W}_{i,j}| \cdot \mathbf{X}_j||_2$) that addresses the input scaling issue that occurs in emergent large magnitude features for LLMs where some features can be 100x times the size of others. This score is very similar to the SparseGPT (Frantar & Alistarh, 2023) metric if only certain elements are retained, but the simplification process in this work drastically reduces the computational requirements. As a result, the authors implement this approach as a single forward-pass across the network and test pruning speed and the resulting inference for a LLaMA-65B model on an NVIDIA A6000.

Regarding the second finding, the authors study several aspects of their approach, known as Wanda, to understand its effectiveness. First, they study the speedup results which they find reduces overhead significantly compared to Frantar & Alistarh (2023). Pruning times are reduced to 0.5 - 5.6 seconds, depending on the size of the LLM pruned. Inference speed is also studied with an average increase of 1.6x over the different linear layers compared to other approaches. The authors also analyze several other aspects of Wanda including, fine-tuning, pruning configurations, robustness to calibration samples, and updates to the weights.

Sharifirad et al. (2025) steps away from the language model and pivots to a different application, a transformer-based model that was utilized to detect anomalies in internal combustion systems. In this paper the authors investigated the execution time on each individual layer of their model, and performed pruning on specific layers based on their L1-score of the execution time of those layers. The authors explored the design space to find that a pruning rate of 30-60% was optimal for three different layers that exhibited this bottleneck behavior. This resulted in a parameter count reduction of 37.2%, execution time reduction by 8.4%, and memory space reduction by 39%.

While the pruning methods discussed above demonstrate impressive compression ratios with minimal accuracy loss over different domains, other work has questioned whether standard evaluation protocols capture the full impact of pruning. Ahia et al. (2021) introduce the concept of the low-resource double bind: communities that most need efficient models (due to compute constraints) often speak low-resource languages that suffer disproportionately from compression. Their experiments on neural machine translation show that pruning preferentially degrades performance on long-tail examples (rare words), unusual constructions, and atypical patterns that require memorization rather than generalization. Crucially, this degradation is masked when evaluation focuses only on frequent, prototypical test examples. The authors demonstrate that a test set capturing long-tail phenomena shows sharper degradation at high sparsity (90%+) than standard evaluation protocols would suggest. This finding has implications for any pruning work targeting deployment in resource-constrained environments.

4.2 Quantization

In neural networks, it is common to utilize 32-bit floating point values to represent the weights and activations of a network. Quantization reduces this precision to some lower level, whether 16-bit, or 8-bit, or lower with a

relatively small impact to the network's accuracy (Nagel et al., 2021). This often takes the form of two broad types of quantization: post-training quantization (PTQ), where the model is trained at a high precision, then truncated or reduced to some lower value. There is also quantization-aware training (QAT) where the model is trained, in whole or part, at the lower precision. When moving from 32 to 8 bits, memory costs are reduced by a factor of 4 and computational costs for the matrix multiplications are reduced by a factor of 16.

These approaches can be insufficient when applied to large language models as explored in Lin et al. (2024). PTQ suffers from large accuracy degradations at low-bit settings and QAT suffers from inefficiency due to its high training cost. The authors observe that edge deployment of large language models is an important goal, but the wide variety of hardware on the market makes this challenge difficult.

In this work, the authors propose a new type of quantization, Activation-aware Weight Quantization (AWQ). AWQ begins with the observation, similar to that of Sun et al. (2024), that not all weights in an LLM are important for inference. However, unlike pruning, where the unimportant weights are removed, AWQ takes the approach of quantizing only these low-importance weights, keeping them at their original precision. Moreover, choosing which weights are unimportant is also examined. The authors investigated reducing the precision of weights through the use of the L_2-norm, but found that this approach was no better than random selection. Instead, selecting salient weights based on *activation value*, which they refer to as **weight quantization** reduced quantization error while also still reducing bit width.

However, implementing this approach, especially for LLMs at the edge, is non-trivial. The authors find that weight access dominates the memory traffic and that a 4-bit integer weight quantization is optimal. The important weights are kept in their original FP16 configuration. The authors then implement on-the-fly weight dequantization where low-precision layers are de-quantized back to FP16 before performing matrix multiplication operations, as the hardware they test on does not have instructions to perform this task. They also apply SIMD-aware weight packing, using platform-specific weight packing aligned to the device's SIMD units.

The authors test this approach on a NVIDIA RTX 4090 w/8GB VRAM, a Jetson Orin (64GB) mobile GPU, and a Raspberry Pi 4, all with LLaMA-2-7B. The 4090 showed an increase from 52 tokens/s to 62 tokens/s, a 3x speed-up for the Jetson Orin, and even the Raspberry Pi-4, which required an extreme INT2 quantization, was able to run a deployed LLM, albeit at 0.7 tokens/s.

Quantization can extend beyond weights and activations. In Elliott et al. (2021) the authors attempt to utilize transformer models in order to perform environmental sound classification on edge hardware. Audio sampling requires a large amount of data and the authors attempted to quantize the training data in order to reduce its size. Multiple audio feature extraction methods were tested which all use the transformer in some way. In particular, their *amplitude reshaping* method reshapes the audio signals where X is a sequence of amplitudes $X = \{x_0, x_1, ..., x_n\}$, l is the sequence length, and d is the hidden dimension

$$X \in \mathbb{R}^{l*d\times 1} \rightarrow_{reshape} X \in \mathbb{R}^{l\times d} \tag{3}$$

which allows the authors to process up to 262,144 samples (6 seconds) of audio. This approach transforms the raw audio from $l * d$ individual samples, impractical to process as individual tokens, into an $l \times d$ matrix where each row is one token composed of the consecutive raw amplitude values.

More appropriate to quantization, however, is the *Curve Tokenization* approach. In this method, the authors note that audio signals are typically smooth and, thus, only have a small number of "curves" that can describe short sequences of audio signals. While the signal is already quantized, the authors further quantize it to reduce the theoretical number of curves that can be in a segment of audio. The authors reduce an 8-token curve to 64% of its normal number of theoretical curves. They continue to tokenize the curves to reduce the sequence length and find that, after creating this quantized vocabulary, between 76-85% of curves in each audio clip in their data set are represented.

In another case, Busia et al. (2025) showcases a vision transformer model that is used to measure heartbeats in order to detect arrhythmia. In their review of related works, the authors determine that transformer-based models produce the best results over the most classes of arrhythmia detection, but the models they studied were too large to fit onto their target platform. In this work, the authors focus on the quantization of the

data and the model down to an 8-bit quantization. Impressively, this paper quantizes both the range of data for each heartbeat detection, but also performs post-training quantization on the non-linear layers of the model, specifically 8-bit integer quantization described in Kim et al. (2021). Their model achieves accuracies comparable to state-of-the-art approaches with 60x fewer parameters and 300x fewer required operations.

4.3 Knowledge Distillation

Knowledge Distillation (KD) was introduced in 2015 in Hinton et al. (2015). This work introduces a student-teacher model relationship, where a large *teacher* model is well trained on a corpus of data. A *student* model, often much smaller than the teacher, is trained using a softened probability distribution of the teacher's logits. The student learns to predict the teacher's outputs, rather than only from the ground truth. This technique, rather than risking the loss of accuracy inherent in pruning and quantization techniques, leverages the teacher's wealth of knowledge to retain the student's accuracy while working with a smaller network. This compression is beneficial for deployment on edge devices as models must have as low of a profile as possible.

Gu et al. (2023) showcases a novel approach to KD for LLMs. Most KD approaches minimize an approximated KL divergence (KLD) in order to force the student's distribution $q_\theta(\mathbf{y}|\mathbf{x})$ to cover all modes of the teacher distribution, $p(\mathbf{y}|\mathbf{x})$. While good for classification tasks, due to the limited number of classes, LLM applications are much more open-ended and complex. $p(\mathbf{y}|\mathbf{x})$ can contain many more modes than what $q_\theta(\mathbf{y}|\mathbf{x})$ can express. As seen in Figure 4, minimizing the forward KLD causes q_θ to assign unreasonably high probabilities to the void regions of p and produces very unlikely samples.

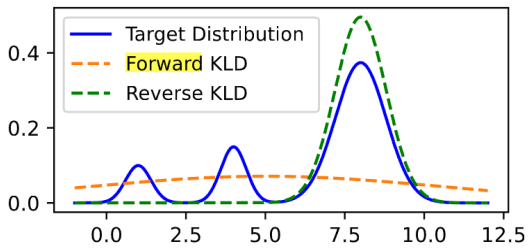

Figure 4: Results from the toy experiment taken from Gu et al. (2023). Graph shows that the Reverse KLD approach more accurately represents the target distribution versus traditional Forward KLD.

Thus, they propose minimization of the *reverse* KLD, which pulls from computer vision and reinforcement learning tasks, and instead assigns low probabilities to the void regions of p. This is because the output spaces in open-ended generative tasks are much more complex and $p(y|x)$ can contain many more modes than what $q_\theta(y|x)$ can express due to the limited model capacity.

MiniLLM was shown to be applicable to a range of model sizes from 120M to 13B parameters. It was not initially clear from their experiments how increasing teacher model sizes improved the student model, but experiments in the paper indicated that the student model follows intuition. Performance of the student model does increase with the size of the teacher model.

4.4 Neural Architecture Search

Neural architecture search (NAS) refers to a set of techniques used to automate the design of deep learning systems (Wistuba et al., 2019). These systems are generally large, complex, and require performance optimization around domain- and hardware-specific constraints. This usually requires significant time and expert input during the design, implementation, and evaluation of deep learning agents. Automating design of deep learning systems can provide considerable reductions in the cost and time to produce performant models, reduce reliance on human expertise in specialized fields, and allow for the exploration of more varied model architecture solutions. In NAS, model design automation is treated as a search space optimization

problem, and as such, it requires one to define a search space, a search strategy, and a performance evaluation metric (Song et al., 2024).

Defining the search space in NAS is a complex problem and has driven much of the research in this field. It requires one to identify which parameters are to be explored. This can include model layer structures and training parameters. More complex search spaces allow for more optimal solutions to be identified at the cost of longer search times and higher computational requirements (Ren et al., 2021), as well as requiring more robust search strategies and evaluation metrics. Early research focused on defining a search space for simpler architectures including feed-forward networks (FFN) and CNNs (Tan et al., 2019; Liu et al., 2019), while more recent works have explored leveraging NAS for generative models (Tong et al., 2025; Wang, 2025). The search strategy is the method used to explore the defined search space, in this case the target elements of the model architecture. Commonly used search strategies include reinforcement learning algorithms, evolutionary algorithms, and gradient-based search algorithms (Song et al., 2024; Hospedales et al., 2022). The performance evaluation metrics are used to quantify the quality of potential solutions considered by the search strategy. These metrics can be used to encourage the selected NAS method to optimize towards things like model accuracy or computational efficiency (Song et al., 2024).

The thought behind automating designs for DNNs in this way is that ML could be leveraged to produce and evaluate architectures more rapidly than humans and may even be able to identify high-performance architectures human designers might never consider. Today, it is commonly used to optimize DNN model performance and reduce their size and computation requirements during operation. It is this latter use case that makes NAS highly applicable to TinyML systems. NAS has historically been used for designing simpler model architectures like CNNs. That trend was driven primarily by a combination of the ease of defining the search space and the relatively low computational requirements of these models. Significant research has recently been conducted to use NAS for transformer-based model design. These models leverage a highly abstract latent space, which can make it difficult to define effective performance metrics and to define an effective search space. Additionally, these models generally have higher training requirements that can make NAS computationally infeasible.

In So et al. (2019), the authors leverage an evolutionary algorithm called tournament selection to define a new transformer architecture. In this work, they define a cell-based search space that separates the encoder and decoder models into stackable cells so that the transformer can be effectively represented and define a set of search parameters derived from an existing transformer-based model. Jing et al. (2020) proposed a generalized NAS method for attention-based models by including attention type as one of the search parameters to allow the optimal attention method to be learned alongside other model parameters. In this paper, Jing *et al.* argue that the optimal behaviors of attention-based models change based on the type of attention being leveraged, so exploring the type of attention in conjunction with the other search parameters will substantially improve the models produced by NAS methods.

Finally, Wang et al. (2020) propose a *Hardware-Aware Transformer* (HAT). In this work, the authors identify two common pitfalls with evaluating transformer-based model efficiency: that FLOPs do not reflect measured latency and that different hardware prefers different transformer architectures. To that end, Wang et al. (2020) train what they refer to as a *SuperTransformer* for machine translation. This large transformer model exists as a super-network that is explored for optimal sub-networks for individual hardware targets as seen in Figure 5.

HAT adds three contributions: the ability to incorporate hardware feedback into model design, low-cost NAS against large design spaces, and certain design insights based on different hardware platforms. For the SuperTransformer, the authors break two common transformer conventions: that the decoder layers of an encoder-decoder transformer design only attend to the last encoder layer and that different layers have the same number of heads, hidden dimensions, and embedding dimensions.

For this work, the authors design a SuperTransformer with the following parameters, which are listed by individual layer width: 512 and 640 embedding dimensions, 1024, 2048, and 3072 hidden dimensions, [4, 8] heads in all attention modules, 6 decoding layers, and QKV vectors fixed at 512. They train their model for 40-50K steps depending on the language task under test. In their results, they identify an insight that models targeted for GPUs prefer wide and shallow designs, while CPU-oriented models prefer narrow and

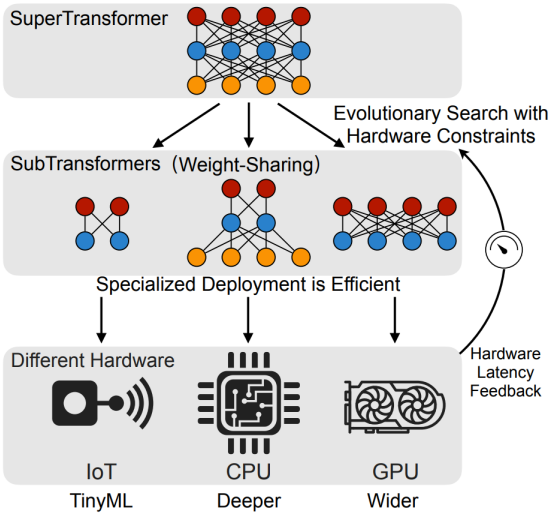

Figure 5: Framework for searching Hardware-Aware Transformers as introduced in Wang et al. (2020). The top frame introduces the SuperTransformer, a large super-network that is explored for optimal SubTransformers. These SubTransformers, seen in the second frame, share certain weights that apply to multiple hardware targets. The bottom frame shows examples of different hardware targets and the different subnetworks that are optimal for each target.

deep designs. All designs are trained together in the SuperTransformer, then HAT selects the top-performing model for the target hardware. This training takes place on cloud environments, but with reduced cost compared to other comparable models. For three separate language translation tasks, HAT produces models with fewer parameters, lower latency, and comparable BLEU scores to other models at comparable training costs. Furthermore, HAT-designed models deployed to the Raspberry Pi achieve state-of-the-art latency scores and comparable BLEU scores to other models.

Finally, HAT is orthogonal to other ML-oriented approaches. HAT uses both 4-bit quantization applied as additional compression as well as knowledge distillation. For the KD approach, HAT uses a high-performance SubTransformer identified through its process, then applies KD to a low-performing one. For this process the teacher-to-student training produced a new network that reduced its BLEU score by 2.4, but with 19M fewer parameters compared to the teacher model.

4.5 Other Approaches

Compression techniques beyond those that are traditionally associated with TinyML were also explored in this survey. For LLMs, there are techniques that try and optimize the Chain-of-Thought (CoT) process. CoT is a methodology that augments LLM processing ability by providing critical capabilities for the decompositions of a task into sub-tasks. The challenge that CoT presents is that the outputs generated by this enhanced reasoning are often much longer than the desired final answers, which increases the inference cost. However, studies indicate that lengthening the steps for CoT reasoning increases an LLM's abilities, and decreasing the steps diminishes its ability to reason.

For example, Kang et al. (2025) investigates this challenge with Conditioned Compressed Chain-of-Thought (C3oT). C3oT is a compression technique for the CoT process that attempts to compress the reasoning steps without reducing accuracy. C3oT begins with a compressor LLM (the authors used GPT-4) that processes input text to condense and retain the core information. This compressor generates two different CoTs for an input text x: a shorter CoT, ${r_i}^{short}$, and a longer CoT ${r_i}^{long}$. This forms elements of a dataset, $D = \{(x_i, {r_i}^{long}, {r_i}^{short}, y_i)\}_{i=1}^{N}$, where x_i is an instruction, y_i is the corresponding answer. The authors then fine-tune LLaMA-2-Chat on D to teach it the relationship between ${r_i}^{long}$ and ${r_i}^{short}$ with the goal of only using ${r_i}^{short}$ during inference. The authors evaluate C3oT using two metrics: accuracy and compression

rate, and compared to other models, C3oT achieves comparable accuracy when compared to the Long CoT methods. This framework can be seen in Figure 6, with the long CoT being fed into the compressor to generate the short CoT which is used to train D and prepare for inference using only ${r_i}^{short}$. The authors compare C3oT to Short CoT and Long CoT, and a version of CoT that uses 100% compression. While performance was promising, the authors also found that more complex reasoning tasks, such as mathematics questions, reduce the ability for the CoT to compress to provide correct answers.

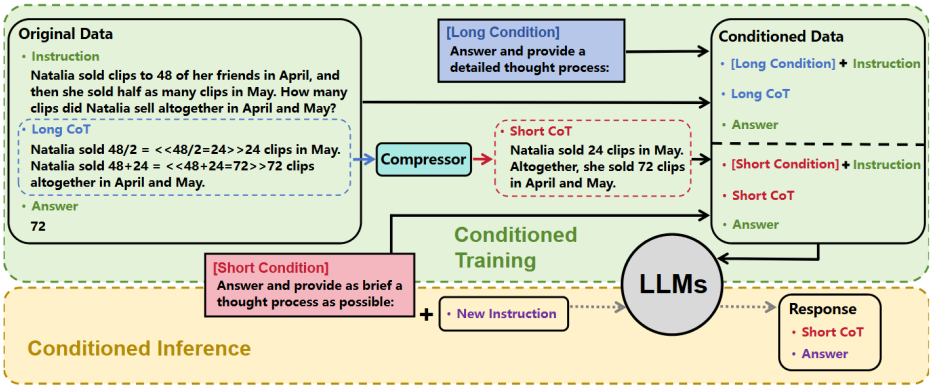

Figure 6: C3oT Framework from Kang et al. (2025).

Additionally, there is a different approach taken by Zhu et al. (2023). Addressing a different need, in that the training for large deep neural networks like transformers is often performed on cloud GPU servers, the authors address this task exclusively. For this, they introduce *PockEngine*, a training engine for edge devices. This training engine adds three innovations: (i) **sparse backpropagation via graph pruning** - a backward computation graph that only updates the most important parameters such as bias-only updates, layer-wise sparse backpropagation, and sub-layer sparse backpropagation, (ii) **compilation-first design** - where the entire training graph is derived at compile-time, rather than at runtime, and (iii) **intermediate representations** - where models defined in common DL libraries like PyTorch are compiled into an intermediate representation that can be targeted for multiple platforms optimally, such as ARM CPUs, the Apple M1, and NVIDIA GPUs.

Finally, Busia et al. (2022) examines a transformer-based model inspired by the vision transformer for seizure detection. This work uses no compression techniques reviewed so far in this survey, rather this research showcases how a transformer model, properly applied to a binary classification task, defies the conventional notion that transformers must be large models while still addressing a requirement of near-zero false positives. The model introduced in this paper uses 8 heads, processing 8-second signal windows on 4 temporal channels and demands only 50.6K parameters at 14.7 MOPS. The result is a model with 99.9% specificity with 0.8 false-positive/hour rate. This result outperforms two different CNN alternatives, both designed and introduced in this same paper, as no prior work on this existed. Finally, this work was deployed to wearable monitoring hardware using an Ambiq Apollo4 MCU on an ARM-Cortex-M4.

5 Co-Design Approaches

TinyML-based and other model compression techniques, as shown in Section 4, focus on designing, compressing and optimizing models for low-resource deployment. However, as noted earlier, hardware plays a role in novel approaches to TinyML application. While hardware-only solutions are not discussed in this survey, software/hardware co-design workflows are worth mentioning. This strategy interprets hardware resource constraints as design parameters. This allows for the creation of novel designs that can take advantage of newer, more optimized hardware. In Sun et al. (2024), we can see this in action. The novel pruning improvements are important conceptually, and comprise the novelty of the paper, but require additional design work in order to truly realize these improvements. The sparse tensor cores in the hardware target allow for the

sparse subset networks to be trained without a radical restructuring of the network. Thus, it makes sense to also explore applications of co-design methodology on transformer models.

TinyFormer (Yang et al., 2023b) is a 3-stage framework that combines multiple approaches for classification tasks. This three-stage design, shown in Figure 7, unites almost every approach discussed in this survey. TinyFormer's first stage, **SuperNAS**, uses defined hardware constraints to form a vast search space and then searches for a model that attempts to balance sparsity and capacity. Then the second stage, **SparseNAS**, selects the best single-path transformer, employs mixed block-wise pruning to improve sparsity and quantizes the model down to 8 bit precision. Finally, a specialized hardware framework, **SparseEngine** is used as the hardware deployment platform. TinyFormer achieves 96.1% accuracy on CIFAR-10 with a tiny budget of 1MB of storage and 320KB of memory.

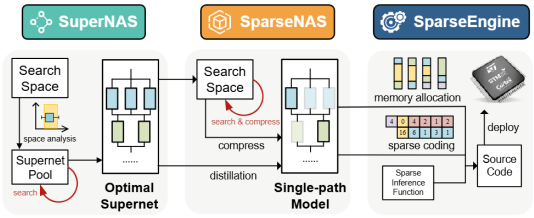

Figure 7: The design of TinyFormer showing its three-stage design: SuperNAS, SparseNAS, and SparseEngine (Yang et al., 2023b).

Co-Design also appears in Song et al. (2025) where the authors use the constraints posed by low-resource devices as design parameters, rather than use existing TinyML techniques that aim to reduce a model's size. In particular, the authors behind SmallThinker begin from a mixture-of-experts design for parameter efficiency. However, the authors combine this approach with a modification to the attention design. In SmallThinker's case, MoE routing, which has replaced the FFN layer, is placed before the attention block instead of after. This has the effect of determining the appropriate expert and allowing prefetching of weights to run in parallel with the attention computation. SmallThinker also uses load balancing for its experts, placing a predictable "hot" subset of experts into DRAM and the remaining "cold" experts into slower storage.

A last co-design approach, though one that leads toward the HW-oriented design workflow, is SwiftTron by Marchisio et al. (2023). SwiftTron is a specialized ASIC accelerator that executes the entire transformer model in integer-only arithmetic by using integer-only approximations for the nonlinear functions employed. SwiftTron creates dedicated hardware units for matrix multiplication, MHA, softmax, FFNs, GELU, and more whith INT8 inputs and INT32 accumulators. This creates hardware-oriented QAT by executing all operations of the transformer in integer-only arithmetic. This is in contrast to Wang et al. (2020) by expressly *avoiding* the quantize/dequantize pattern. This co-design solution is synthesized in 65 nm CMOS technology.

6 Deployment to Low-Resource Devices

It is important not only to examine training and development of transformer-based models, but also to examine deployment of models as well. As many types of machine learning models currently exist today and are useful for many varied tasks, understanding some efforts to deploy models to low-resource devices may be useful. Previous sections have examined both transformer model compression techniques and co-designs of transformer architectures with custom hardware designs. This section will examine a couple of deployment strategies that leverage TinyML techniques.

The first paper examined, while not using the transformer model as a base is worth mentioning. In Han et al. (2024), the authors pose the problem of running convolutional neural networks on severely constrained resource devices. Traditional pruning methods either reduce accuracy or inference speed beyond acceptable levels. The authors provide three novelties to address this challenge: (i) a weight compressed storage data structure that position, (ii) reducing index overhead, (iii) a custom convolution operator that uses SIMD

parallelism for fast inference while avoiding idle clock cycles, and (iv) a pruning strategy scheduler that determines a per-layer pruning ratio as an optimization problem.

This approach is important to briefly highlight, though it does not address transformer directly, to highlight how multiple pruning approaches can be used simultaneously in novel designs. Moreover, it also shows how it is important to understand approaches that can be applied to existing models available today. Han et al. (2024) can be integrated into TensorFlow Lite Micro and only adds 4KB of overhead, making it useful for developers without access to AI services. Comparatively, Jung et al. (2025) examines this same problem, deploying models to extremely low-resource devices, from the transformer perspective.

The authors of this paper are aware of the promise of transformer-based models to outpace more traditional approaches in accuracy. The MHA introduces challenges, however, that these traditional approaches weren't designed to handle, namely high memory peaks, expensive data reshaping, and parallelization scaling. This work does not leverage any of the TinyML operations mentioned in this survey, but provides three contributions. First, is an optimized MHA library which is tailored to minimize overhead associated with data marshaling operations and maximize data reuse for each attention layer. Second, are two novel optimizations: a fusion-based tiling scheme for MHA operations and an offline weight fusion schedule for the MHA operations. Combined, these optimizations reduce memory peak by 6.19x, latency by 1.53x, and parameter count by 25%. Finally, the authors benchmark their libraries and optimizations using published transformers.

7 Analyzing Trends in Technique/Hardware Distributions

Over the last 6 years, there has been incredible growth in studies for low-resource solutions and applications of transformer-based models. This trend reflects a growing interest in applying this powerful model to the edge. However, as shown in Figure 8, the number of studies in this intersection is still very low. Systematic searches through databases may miss works that fall under traditional search terms, though they provide novelty in this area. Even so, the lack of available research is significant, but some patterns do emerge in the works that are available. Our narrow focus on consumer-targeted hardware reflects trends that can be also seen in the larger corpus of works first identified in our search. Figure 9 shows these same trends, but from the full list of 755 works first algorithmically categorized. Like our review of our focused papers, Internet of Things (IoT) devices still dominate the focus of TinyML-oriented research.

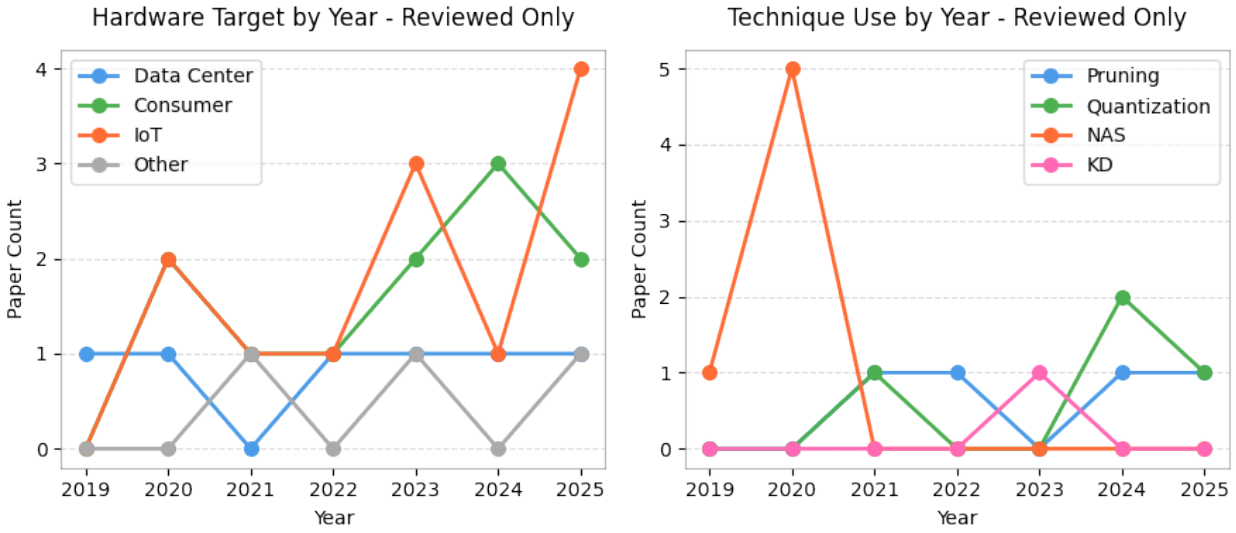

Figure 8: Trends of works reviewed in this survey by hardware targets and techniques.

Our curated list was chosen to analyze the various techniques as applied to this consumer-oriented approach, which can be seen in Figure 8. While we discovered that the distribution of compression techniques was

nearly even, the small sample size curated by our desired hardware target makes it difficult to support this claim. Therefore, we expanded our analysis to the entire set of works first identified in Google Scholar to see if the trends seen in the small set still hold with a larger sample. Expanding our search shows that KD, which is nearly absent in our list, is still frequently the smallest represented technique in the full set for most of the years surveyed. This indicates that knowledge distillation as a compression technique has historically been under-represented in both the consumer and TinyML spaces, although 2025 saw a marked increase in KD papers available.

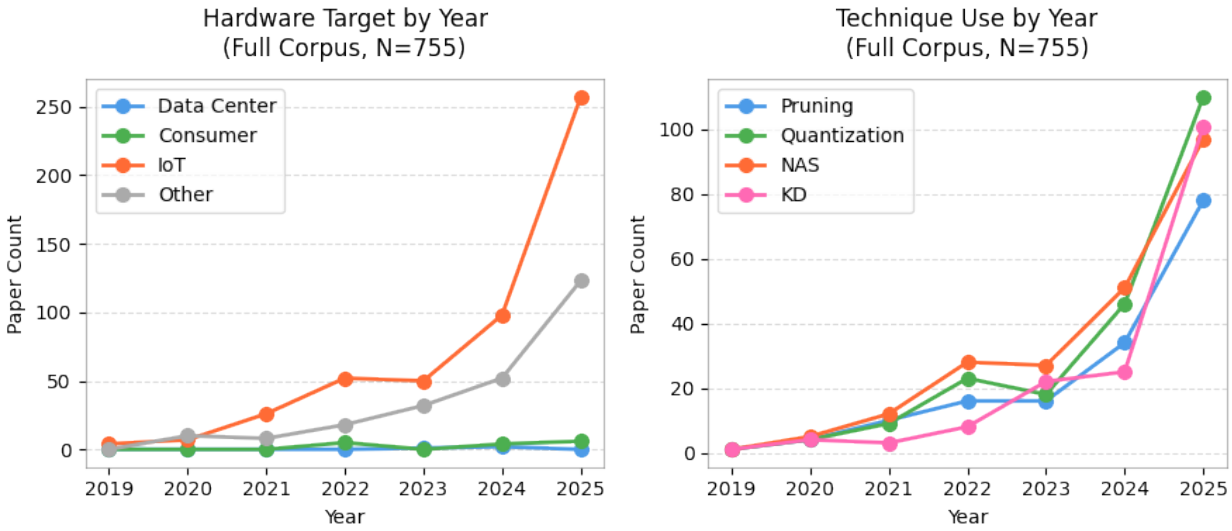

Figure 9: Research trends for hardware targets and technique usage, using search terms from Table 1. This figure represents the entire corpus of 755 works retrieved from Google Scholar.

For hardware, the trends do change from the sample to the full set. IoT, as mentioned earlier, dominates in both the small sample and the full set, but Other works, curated out of this review for a focus on consumer targets sits firmly as the second most common technique. Table 3 shows how we defined each of the targets in this survey. Many of the targets that fall under Other are technically IoT by compute requirements, but fail to meet the consumer-focused scope we target in this survey. For the entire corpus, Consumer targets are nearly absent with our curated set adding additional works not initially found in Google Scholar to be able to review. This is a major gap of targets that are not large data centers nor tiny MCUs.

Hardware Category	**Definition**
IoT	MCUs and low-cost, low-resource edge devices generally available to consumers
Consumer	CPUs/GPUs generally available to consumers
Data Center	Enterprise GPUs and other hardware not available to consumers
Other	Targets that do not fit in the other three categories

Table 3: Hardware categories used in this survey.

Moreover, Figure 8 does not reference specific hardware design choices that the various papers leveraged to support their results, though its novelty may have been one specific technique. Indeed, Sun et al. (2024) seems to imply much of its success is due *to* the specific hardware that its deployment target possesses. Therefore, it is important to examine not only software techniques, but also optimize transformer models on edge hardware, but also look at the hardware itself. Also, it is clear that NAS is the dominant technique on display. 2020 features several works where NAS was the primary technique showcased. This agrees with results presented in several studies that claim that only a small subset of weights truly matter in large transformer models. What is curious is that knowledge distillation as a technique is very under-represented in terms of novelty. Although Wang et al. (2020) does implement KD as part of their results, the authors

mention that KD is orthogonal to their novelty. Moreover, most of the novelty introduced in the surveyed works use additional techniques orthogonally, adding them as footnotes in their workflow design.

Also interesting is the distribution of techniques against the deployment platform. Figure 10 shows each technique surveyed against the hardware it was deployed on for the full corpus of papers from Google Scholar. Unsurprisingly, IoT has the largest count, but, in terms of techniques, leans into the "Other" category. In many of these papers, the hardware target was not identified. This may indicate that only applying classical TinyML techniques may not be sufficient in completely pushing transformers to the edge. For application targets, many of the works in this survey aimed to improve the language model, either in machine translation or the large language model. Detection and classification tasks were also featured, across diverse fields such as medical, anomaly detection, and environmental sound classification.

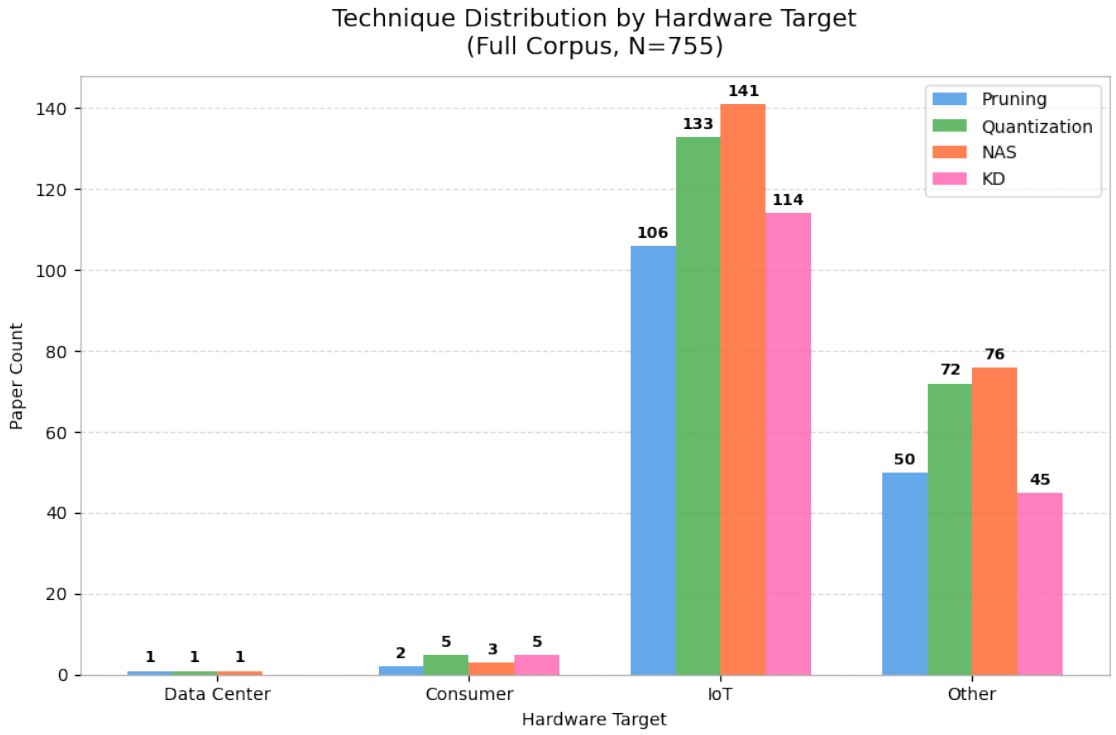

Figure 10: Chart showing the distribution of techniques against their targeted hardware. For TinyML, known IoT targets clearly dominate, with Other targets in second place.

8 Conclusion and Future Efforts

Transformer-based models have shown incredible promise for state-of-the-art results in many applications (Lin et al., 2022b). However, the resource demands of transformer models discourage application of these powerful tools in edge or low-resource environments. The works cited in this survey show that TinyML techniques can be successfully applied to transformer architectures across a diverse range of hardware targets and application domains.

A common pattern across the surveyed works is that the techniques are largely orthogonal to each other. Papers that introduce novelty do not typically combine their contributions with other papers, such as pruning and knowledge distillation, for instance. Some works do apply multiple techniques together and note that their contributions are orthogonal to others, but that combination space remains almost entirely unexplored. This means that the existing literature is better understood as a collection of independent building blocks rather than competing solutions to the same problem. The low-hanging fruit for progress in this area lies in assembling these components.

Within that combination space, three gaps stand out. First, knowledge distillation is the most underrepresented technique relative to its potential value. Of the nineteen surveyed works, only one identifies KD as its primary contribution, despite KD being one of the canonical TinyML techniques and arguably the one best suited to the scale of transformer models. Second, most works in this survey address inference only; only two involved novel training approaches. Edge devices that cannot train locally are brittle when encountering data distributions that differ from their training set, and local training remains an open problem. Third, the combination of techniques itself has received almost no systematic study.

Future work that combines the orthogonal contributions already present in the literature, particularly KD combined with pruning or quantization, and training frameworks combined with efficient architecture designs — represents the most accessible path toward truly capable transformer models at the edge. Based on the findings of this survey, future research directions could focus on several areas: training approaches for TinyML/Transformers, novel approaches to the TinyML techniques provided here, especially in knowledge distillation, combinations of techniques to improve performance or reduce memory requirements, and additional applications for this intersection.

Acknowledgments

I would like to thank .....

References

Youssef Abadade, Anas Temouden, Hatim Bamoumen, Nabil Benamar, Yousra Chtouki, and Abdelhakim Senhaji Hafid. A comprehensive survey on tinyml. IEEE Access, 11:96892–96922, 2023. doi: 10.1109/ACCESS.2023.3294111. 2

Orevaoghene Ahia, Julia Kreutzer, and Sara Hooker. The low-resource double bind: An empirical study of pruning for low-resource machine translation. In Findings of the Association for Computational Linguistics: EMNLP 2021, pp. 3316–3333. Association for Computational Linguistics, 2021. URL https://aclanthology.org/2021.findings-emnlp.282. 7, 8

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473, 2014. 3

Paola Busia, Andrea Cossettini, Thorir Mar Ingolfsson, Simone Benatti, Alessio Burrello, Moritz Scherer, Matteo Antonio Scrugli, Paolo Meloni, and Luca Benini. Eegformer: Transformer-based epilepsy detection on raw eeg traces for low-channel-count wearable continuous monitoring devices. In 2022 IEEE Biomedical Circuits and Systems Conference (BioCAS), pp. 640–644, 2022. doi: 10.1109/BioCAS54905.2022.9948637. 7, 13

Paola Busia, Matteo Antonio Scrugli, Victor Jean Baptiste Jung, Luca Benini, and Paolo Meloni. A Tiny Transformer for Low-Power Arrhythmia Classification on Microcontrollers. IEEE Transactions on Biomedical Circuits and Systems, 19(1):142–152, 2025. ISSN 19409990. doi: 10.1109/TBCAS.2024.3401858. 7, 9

Luigi Capogrosso, Federico Cunico, Dong Seon Cheng, Franco Fummi, and Marco Cristani. A Machine Learning-oriented Survey on Tiny Machine Learning. IEEE Access, 12:23406–23426, 2024. 2, 4, 5

Misha Denil, Babak Shakibi, Laurent Dinh, Marc'Aurelio Ranzato, and Nando De Freitas. Predicting parameters in deep learning. Advances in neural information processing systems, 26, 2013. 6

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers), pp. 4171–4186, 2019. 2, 3

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. arXiv preprint arXiv:2010.11929, 2020. 4, 5

David Elliott, Carlos E Otero, Steven Wyatt, and Evan Martino. Tiny transformers for environmental sound classification at the edge. arXiv preprint arXiv:2103.12157, 2021. 7, 9

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. arXiv preprint arXiv:1803.03635, 2018. 6

Elias Frantar and Dan Alistarh. Sparsegpt: massive language models can be accurately pruned in one-shot. In Proceedings of the 40th International Conference on Machine Learning, ICML'23. JMLR.org, 2023. 8

Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. Advances in neural information processing systems, 27, 2014. 2

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. Minillm: Knowledge distillation of large language models. arXiv preprint arXiv:2306.08543, 2023. 7, 10

Lixiang Han, Zhen Xiao, and Zhenjiang Li. Dtmm: Deploying tinyml models on extremely weak iot devices with pruning. In IEEE INFOCOM 2024-IEEE Conference on Computer Communications, pp. 1999–2008. IEEE, 2024. 7, 14, 15

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149, 2015. 7

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531, 2015. 6, 10

Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. Meta-learning in neural networks: A survey. IEEE Transactions on Pattern Analysis and Machine Intelligence, 44(9):5149–5169, 2022. doi: 10.1109/TPAMI.2021.3079209. 11

JarvisLabs. Nvidia h100 price guide 2026. Available: https://docs.jarvislabs.ai/blog/h100-price#llm-training-cost-analysis, 2026. (accessed on 2026-01-08). 2

Kun Jing, Jungang Xu, and Hui Xu Zugeng. Nasabn: A neural architecture search framework for attention-based networks. In 2020 International Joint Conference on Neural Networks (IJCNN), pp. 1–7, 2020. doi: 10.1109/IJCNN48605.2020.9207600. 7, 11

Victor Jean-Baptiste Jung, Alessio Burrello, Moritz Scherer, Francesco Conti, and Luca Benini. Optimizing the deployment of tiny transformers on low-power mcus. IEEE Transactions on Computers, 74(2):526–541, 2025. doi: 10.1109/TC.2024.3500360. 7, 15

Yu Kang, Xianghui Sun, Liangyu Chen, and Wei Zou. C3ot: Generating shorter chain-of-thought without compromising effectiveness. Proceedings of the AAAI Conference on Artificial Intelligence, 39(23): 24312–24320, Apr. 2025. doi: 10.1609/aaai.v39i23.34608. URL https://ojs.aaai.org/index.php/AAAI/article/view/34608. 7, 12, 13

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. arXiv preprint arXiv:2001.08361, 2020. 3

Sehoon Kim, Amir Gholami, Zhewei Yao, Michael W. Mahoney, and Kurt Keutzer. I-bert: Integer-only bert quantization. In Marina Meila and Tong Zhang (eds.), Proceedings of the 38th International Conference on Machine Learning, volume 139 of Proceedings of Machine Learning Research, pp. 5506–5518. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/kim21d.html. 10

Mikhail V Koroteev. Bert: a review of applications in natural language processing and understanding. arXiv preprint arXiv:2103.11943, 2021. 3

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger (eds.), Advances in Neural Information Processing Systems, volume 25. Curran Associates, Inc., 2012. URL https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf. 1

Nir Kshetri. The environmental impact of artificial intelligence. IT Professional, 26(03):9–13, 2024. 1

Woosuk Kwon, Sehoon Kim, Michael W. Mahoney, Joseph Hassoun, Kurt Keutzer, and Amir Gholami. A fast post-training pruning framework for transformers. Advances in Neural Information Processing Systems, 35:24101–24116, 10 2022. URL https://arxiv.org/abs/2204.09656. 7

Ji Lin, Ligeng Zhu, Wei-Ming Chen, Wei-Chen Wang, Chuang Gan, and Song Han. On-device training under 256kb memory. Advances in Neural Information Processing Systems, 35:22941–22954, 2022a. 2

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. Proceedings of machine learning and systems, 6:87–100, 2024. 7, 9

Tianyang Lin, Yuxin Wang, Xiangyang Liu, and Xipeng Qiu. A survey of transformers. AI open, 3:111–132, 2022b. 17

Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan L. Yuille, and Li Fei-Fei. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 82–92, 2019. doi: 10.1109/CVPR.2019.00017. 11

Alberto Marchisio, Davide Dura, Maurizio Capra, Maurizio Martina, Guido Masera, and Muhammad Shafique. Swifttron: An efficient hardware accelerator for quantized transformers. In 2023 International Joint Conference on Neural Networks (IJCNN), pp. 1–9, 2023. doi: 10.1109/IJCNN54540.2023.10191521. 7, 14

Markus Nagel, Marios Fournarakis, Rana Ali Amjad, Yelysei Bondarenko, Mart Van Baalen, and Tijmen Blankevoort. A white paper on neural network quantization. arXiv preprint arXiv:2106.08295, 2021. 6, 9

Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-yao Huang, Zhihui Li, Xiaojiang Chen, and Xin Wang. A comprehensive survey of neural architecture search: Challenges and solutions. ACM Comput. Surv., 54 (4), May 2021. ISSN 0360-0300. doi: 10.1145/3447582. URL https://doi.org/10.1145/3447582. 11

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 10684–10695, 2022. 1, 2

Shaibal Saha and Lanyu Xu. Vision transformers on the edge: A comprehensive survey of model compression and acceleration strategies. Neurocomputing, pp. 130417, 2025. 5

Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlicket, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, et al. BLOOM: A 176B-Parameter Open-Access Multilingual Language Model. In BigScience Workshop. CoRR, 2022. URL https://doi.org/10.48550/arXiv.2211.05100. 1, 2, 3

Sandeep Singh Sengar, Affan Bin Hasan, Sanjay Kumar, and Fiona Carroll. Generative artificial intelligence: a systematic review and applications. Multimedia Tools and Applications, 84(21):23661–23700, 2025. 2

Iman Sharifirad, Jalil Boudjadar, and Peter Gorm Larsen. Tinyml for computation-aware transformer-based anomaly detection in internal combustion systems. In 2025 IEEE 23rd World Symposium on Applied Machine Intelligence and Informatics (SAMI), pp. 000141–000146, 2025. doi: 10.1109/SAMI63904.2025.10883083. 7, 8

David R. So, Chen Liang, and Quoc V. Le. The evolved transformer. In International Conference on Machine Learning, 2019. URL https://api.semanticscholar.org/CorpusID:59523610. 7, 11

Xiaotian Song, Xiangning Xie, Zeqiong Lv, Gary G. Yen, Weiping Ding, Jiancheng Lv, and Yanan Sun. Efficient evaluation methods for neural architecture search: A survey, 2024. URL https://arxiv.org/abs/2301.05919. 11

Yixin Song, Zhenliang Xue, Dongliang Wei, Feiyang Chen, Jianxiang Gao, Junchen Liu, Hangyu Liang, Guangshuo Qin, Chengrong Tian, Bo Wen, et al. Smallthinker: A family of efficient large language models natively trained for local deployment. arXiv preprint arXiv:2507.20984, 2025. 7, 14

Mingjie Sun, Zhuang Liu, Anna Bair, and J. Zico Kolter. A simple and effective pruning approach for large language models, 2024. URL https://arxiv.org/abs/2306.11695. 7, 8, 9, 13, 16

Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. Mnasnet: Platform-aware neural architecture search for mobile. In 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 2815–2823, 2019. doi: 10.1109/CVPR.2019.00293. 11

Lyuyang Tong, Jie Liu, and Bo Du. Sceneformer: Neural architecture search of transformers for remote sensing scene classification. IEEE Transactions on Geoscience and Remote Sensing, 63:1–15, 2025. doi: 10.1109/TGRS.2025.3555207. 11

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. Advances in neural information processing systems, 30, 2017. 1, 2, 3

Hanrui Wang, Zhanghao Wu, Zhijian Liu, Han Cai, Ligeng Zhu, Chuang Gan, and Song Han. HAT: Hardware-aware transformers for efficient natural language processing. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault (eds.), Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pp. 7675–7688, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.686. URL https://aclanthology.org/2020.acl-main.686/. 7, 11, 12, 14, 16

Shang Wang. Elastic architecture search for efficient language models. In 2025 IEEE International Conference on Multimedia and Expo (ICME), pp. 1–6, 2025. doi: 10.1109/ICME59968.2025.11210235. 11

Zach Warren, Mike Abbot, Lucy Leach, Steve Seemer, Molly Seymour, Tiny Tayor, and Gregg Wirth. 2025 Generative AI in Professional Services Report. In Thomson Reuters Institute, 2025. 1

Martin Wistuba, Ambrish Rawat, and Tejaswini Pedapati. A survey on neural architecture search. CoRR, abs/1905.01392, 2019. URL http://arxiv.org/abs/1905.01392. 10

Jianlei Yang, Jiacheng Liao, Fanding Lei, Meichen Liu, Junyi Chen, Lingkun Long, Han Wan, Bei Yu, and Weisheng Zhao. TinyFormer: Efficient Transformer Design and Deployment on Tiny Devices. arXiv preprint arXiv:2311.01759, 2023a. 3

Jianlei Yang, Jiacheng Liao, Fanding Lei, Meichen Liu, Junyi Chen, Lingkun Long, Han Wan, Bei Yu, and Weisheng Zhao. Tinyformer: Efficient transformer design and deployment on tiny devices. arXiv preprint arXiv:2311.01759, 2023b. 7, 14

Ligeng Zhu, Lanxiang Hu, Ji Lin, Wei-Ming Chen, Wei-Chen Wang, Chuang Gan, and Song Han. Pockengine: Sparse and efficient fine-tuning in a pocket. In Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO '23, pp. 1381–1394, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400703294. doi: 10.1145/3613424.3614307. URL https://doi.org/10.1145/3613424.3614307. 7, 13

Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. arXiv preprint arXiv:1710.01878, 2017. 8