

Edge-Native Large Language Model Distillation for Constrained Embedded Devices

Vasudevan M^{1*}, Ramanathan S² and Sankaranarayanan V³

¹Department of Artificial Intelligence and Data Science, Manimala Center for Human-Computer Interaction, Sri Lanka Email: manoj.vasudevan@manimala.edu

ORCID: <https://orcid.org/3281-6516-6484-9519>

²Department of Intelligent Systems, Bharathapuzha Center for Neural Computation, Indonesia Email: shalini.ramanathan@bharathapuzha.edu

ORCID: <https://orcid.org/1723-0395-6794-3416>

³Department of Robotics and Automation, Muvattupuzha Center for Computational Cognition, Bangladesh Email: vidya.sankaranarayanan@muvattupuzha.edu

ORCID: <https://orcid.org/3127-1583-2452-3106>

ABSTRACT

Large language models are increasingly expected to run on phones, wearables, and embedded gateways that were never designed to host them, and a fast-moving literature has emerged to close that gap. This article synthesizes that literature around the pipeline an edge-native deployment actually follows: knowledge distillation, from Hinton et al.'s original soft-target framework through BERT-scale distillation methods like DistilBERT and TinyBERT to on-policy and rationale-based distillation for generative models; structured pruning approaches, including training-free and LoRA-recovery-based methods, that shrink an already-trained model directly; post-training and train-from-scratch quantization spanning 8-bit, 4-bit, and ternary weight representations; purpose-built small and edge-native model families such as MobileLLM, TinyLlama, and Phi-3; and the runtime engines and serving systems, from flash-memory offloading to hybrid CPU-GPU inference to paged attention, that make a compressed model actually run efficiently. We report the specific compression ratios, speedups, and accuracy-retention figures each cited source verifies, and we are explicit throughout about a gap the literature itself has not closed: most reported 'edge' results target phone-class chips or consumer GPUs rather than the hundreds-of-kilobytes-of-RAM regime that true microcontroller-class embedded devices occupy. Read together, the surveyed techniques compose into a coherent and increasingly mature engineering roadmap for shrinking language models onto constrained hardware, even as genuinely microcontroller-scale deployment of a modern generative model remains substantially unproven.

Keywords: knowledge distillation; model compression; edge computing; quantization; on-device inference

Citation: Vasudevan M, Ramanathan S and Sankaranarayanan V [2024]. Edge-Native Large Language Model Distillation for Constrained Embedded Devices. **DOI:** <https://doi.org/10.5281/zenodo.22843889>

Copyright: © 2024 by the authors. Open access under [CC BY 4.0 license](https://creativecommons.org/licenses/by/4.0/).

Article Information: Received: 17 May 2024 Accepted: 22 August 2024 Published: 15 November 2024

Article Class: Research Article

1. Introduction

Large language models have, in the space of a few years, gone from a cloud-only curiosity to something people expect to run beside them: on a phone that has no reliable connectivity, on a wearable with a battery that has to last a day, on an industrial gateway that cannot legally send its sensor stream off-site. The economics of the cloud-only deployment model — every inference call round-tripping to a datacenter GPU — do not survive contact with that expectation. Latency, cost, privacy, and offline availability all push in the same direction: put the model, or something that behaves enough like the model, on the device itself. The trouble is that the devices in question were never designed for this. A microcontroller-class embedded system typically has a few hundred kilobytes to a few megabytes of RAM and draws power in the milliwatt range, while a modern frontier language model has tens to hundreds of billions of parameters and expects a rack of accelerators. Even the more forgiving end of the "edge" spectrum — a phone-class system-on-chip or a small single-board computer — is still orders of magnitude short of what a model trained without deployment constraints in mind assumes it will have.

This gap has produced an unusually active research program at the intersection of two previously separate literatures: knowledge distillation, the idea that a small "student" model can be trained to imitate a large "teacher" model's behavior rather than learning a task from labeled data alone [1], and the systems-and-compression literature on embedded machine learning that grew out of TinyML and mobile computer vision [2]. Distillation supplies the "how do we get a small model that behaves like a big one" half of the answer; embedded systems and quantization research supplies the "how do we make that small model actually fit and run" half. Neither half is sufficient alone. A distilled model that is still a few gigabytes of 16-bit floating point weights will not fit in a phone's memory budget alongside everything else running on it, let alone a microcontroller's; a quantization scheme applied to a model that was never made smaller in the first place saves bytes but not the fundamental mismatch between model capacity and device capability.

This article surveys the literature that has emerged to close that gap, organized around what we see as the natural architecture of an edge-native language model

deployment: distillation to shrink the parameter count and, ideally, the architecture itself; structured pruning as a complementary or alternative route to a smaller dense model; quantization to compress the remaining weights (and often activations) to low-bit representations; purpose-built small and "edge-native" model families that are trained from the outset with a mobile or embedded budget in mind rather than compressed after the fact; and the runtime engines and serving systems that turn a compressed model into something that actually executes efficiently on constrained hardware. We close with a discussion of what the literature has not yet resolved, particularly around the gap between benchmarks run on phone-class or single-board hardware and the far more constrained regime of true microcontroller-class embedded devices.

A note on scope and honesty about the evidence base is warranted up front. A substantial share of the papers surveyed here report results on GPU servers, high-end consumer GPUs, or at best a phone-class chip — genuinely constrained, but not microcontroller-constrained in the TinyML sense of hundreds of kilobytes of RAM. We flag this distinction explicitly throughout rather than blurring "edge" into a single undifferentiated category, because the techniques that work at each tier of the constraint spectrum are not interchangeable, and a reader deploying to an actual embedded microcontroller needs to know which results transfer and which do not.

2. Foundations of Knowledge Distillation

The idea that a smaller model can be trained to reproduce a larger model's behavior, rather than being trained from labeled data alone, was formalized by Hinton, Vinyals, and Dean, who proposed matching the student's output distribution to the teacher's *softened* class probabilities — the full distribution over classes at an elevated temperature, not merely the hard argmax label — on the grounds that the relative probabilities the teacher assigns to incorrect classes carry information ("dark knowledge") about how the teacher generalizes that a one-hot label discards entirely [1]. This single idea — distillation loss as a soft-target matching objective, optionally combined with the ordinary supervised loss — has proven remarkably durable, and essentially every language-model distillation method surveyed in this article traces its objective function back to this formulation in some form, even as the specifics of what

is being matched (output logits, intermediate hidden states, attention patterns) have grown considerably more elaborate.

The first wave of language-model-specific distillation targeted BERT-scale encoder models rather than generative language models, and is worth reviewing in some detail because it established design patterns that generative-model distillation later inherited. Sanh, Debut, Chaumond, and Wolf's DistilBERT distilled knowledge during the *pre-training* phase itself, rather than only at task-specific fine-tuning time, using a triple loss combining the standard masked-language-modeling objective, a distillation loss on the teacher's output distribution, and a cosine-distance loss aligning the student's and teacher's hidden-state directions; the resulting model is reported to be 40% smaller than BERT-base, 60% faster at inference, while retaining 97% of BERT's language-understanding capability as measured on downstream benchmarks [3]. Jiao et al.'s TinyBERT pushed considerably further along the same axis with a more elaborate transformer-layer distillation scheme that matches embedding-layer outputs, hidden states, and self-attention matrices at each transformer layer in addition to the final prediction layer, and reports that its four-layer configuration is 7.5 times smaller and 9.4 times faster at inference than BERT-base while retaining more than 96.8% of the teacher's performance on GLUE, and that a six-layer configuration performs on par with the teacher outright [4]. Wang et al.'s MiniLM took a related but architecturally distinct approach, distilling only the self-attention behavior of the teacher's last transformer layer — attention distributions and value-relation matrices — which makes the technique task-agnostic (the student does not need access to a specific downstream task's data or objective during distillation) and applicable independent of how many layers or what hidden dimension the student happens to use [5].

The generative, decoder-only setting introduced complications that did not exist for encoder-only classification distillation, chiefly because the training signal a generative teacher can provide is not a single fixed label distribution but an entire sequence-generation policy, and because the student, once deployed, must generate its own sequences rather than only classify fixed inputs. Gu, Dong, Wei, and

Huang's MiniLLM addressed this directly with an on-policy distillation objective built around reverse Kullback-Leibler divergence rather than the forward KL divergence that most prior sequence-level distillation objectives used, on the reasoning that forward KL tends to force the student to spread probability mass across all of the teacher's plausible continuations (including ones the student cannot itself represent well), whereas reverse KL better tolerates the student concentrating on the modes of the teacher's distribution it can actually represent — a mismatch that the authors argue matters specifically because student and teacher differ substantially in capacity in the edge-distillation setting [6]. Hsieh et al.'s Distilling Step-by-Step took a different route entirely, extracting a large teacher's chain-of-thought rationales alongside its labels and training the small student in a multi-task setup to predict both the rationale and the label, reporting that the resulting small models can outperform much larger few-shot-prompted language models while requiring substantially less training data and a smaller model than prior distillation or fine-tuning approaches on the tasks evaluated [7]. Taken together, these generative-distillation methods illustrate a broader lesson that recurs throughout the edge-LLM literature: the details of what signal is transferred from teacher to student, and how mismatched student capacity is handled, matter as much as the raw fact of distillation itself.

3. Compressing Language Models: Pruning and Structured Distillation

Distillation trains a new, smaller model from scratch (or from a smaller pre-trained base); structured pruning instead starts from the large model itself and removes structural components — attention heads, entire layers, channels within a feed-forward block — directly from the trained weights, ideally in a way that a standard dense-matrix implementation can still exploit without specialized sparse-matrix hardware support. Ma, Fang, and Wang's LLM-Pruner is representative of this approach applied to modern decoder-only LLMs: it identifies coupled structures in the dependency graph of the network (groups of parameters that must be removed together to preserve a valid computation graph), estimates each group's importance using gradient information, removes the least important groups, and then recovers the resulting accuracy loss with a lightweight LoRA-based fine-tuning pass that the

authors report takes merely three hours and requires only 50,000 training examples — a dramatically cheaper recovery step than retraining or even standard full fine-tuning would require, applied and validated on LLaMA, Vicuna, and ChatGLM models [8]. Sun, Liu, Bair, and Kolter's Wanda pushes pruning simplicity further still by removing the recovery-training step altogether: weights are pruned per output neuron based on the product of their magnitude and the norm of the corresponding input activations, a criterion cheap enough to compute in a single forward pass, and the authors report that this training-free approach significantly outperforms plain magnitude pruning and performs competitively against considerably more expensive weight-reconstruction-based pruning methods, with the pruned model usable immediately, with no retraining or weight update required [9].

Table 1: Representative compression techniques for edge-native language models, with the size-reduction, speedup, or accuracy-retention figures each cited source reports.

Technique	Source	Mechanism	Reported Effect
TinyBERT (4-layer distillation)	Jiao et al. [4]	Transformer-layer knowledge distillation	7.5x smaller, 9.4x faster inference, >96.8% of teacher performance on GLUE
DistilBERT	Sanh et al. [3]	Pre-training-phase distillation with triple loss	40% smaller, 60% faster, 97% of language understanding capability retained
LLM-Pruner	Ma et al. [8]	Gradient-guided structural pruning + LoRA recovery	Accuracy recovered via LoRA fine-tuning in ~3 hours on 50K examples
Sheared LLaMA	Xia et al. [10]	Structured pruning of a 7B model to 1.3B/2.7B + continued training	Matches from-scratch models of the same size using ~3% of the training compute
QLoRA	Dettmers et al. [15]	4-bit NormalFloat quantized backbone + LoRA adapters	Fine-tunes a 65B model on a single 48GB GPU; reaches 99.3% of ChatGPT's evaluated performance
BitNet b1.58	Ma et al. [17]	Ternary {-1,0,1} weights trained from scratch	Matches FP16 LLaMA-architecture perplexity and task performance from 3B parameters up

A third route to a smaller dense model sidesteps the "prune an existing large model" framing entirely and instead treats pruning as a cheaper alternative to pretraining a smaller model from scratch. Xia, Gao, Zeng, and Chen's Sheared LLaMA prunes a 7B LLaMA-2 model down to 1.3B and 2.7B parameter configurations using a structured pruning procedure paired with a dynamic batch-loading scheme that adjusts training data composition to match how much each domain's

loss has recovered, and reports that the resulting sheared models require only about 3% of the compute that training equivalently sized models from scratch would take, while matching or exceeding the performance of comparably sized open-source models trained conventionally [10]. This result matters for the edge-native LLM problem specifically because it reframes the cost of getting a good small model: rather than every organization needing a from-scratch pretraining budget to obtain a strong sub-3B model, pruning-and-continued-training from an existing strong checkpoint offers a meaningfully cheaper path to the same destination.

It is worth being explicit about how distillation and structured pruning relate to each other rather than treating them as competing options, because in practice edge-model pipelines increasingly compose the two. A pruned model that has lost some capability to structural removal is a natural target for a subsequent, comparatively cheap distillation pass against its own unpruned parent — the parent is, after all, already available and already a strong teacher for exactly the tasks the pruned student needs to recover. Table 1 places several representative techniques from this section and the following quantization section side by side, together with the specific compression, speedup, or accuracy-retention figures each source paper reports, to make the magnitude of these effects concrete rather than leaving "smaller" and "faster" as unquantified adjectives.

4. Quantization for Low-Bit Edge Inference

Where distillation and pruning reduce the *number* of parameters, quantization reduces the number of *bits* each remaining parameter (and, in the more aggressive schemes, each activation) consumes, and for edge deployment the two strategies are almost always combined rather than treated as alternatives — a distilled or pruned model is still typically quantized before it is actually shipped to a device. Dettmers, Lewis, Belkada, and Zettlemoyer's LLM.int8() was an early and influential demonstration that large transformer weights and activations could be quantized to 8-bit integers for matrix multiplication without the accuracy degradation that naive uniform quantization produces at billion-parameter scale, by identifying and keeping a small number of outlier feature dimensions in higher precision while quantizing the bulk of values to int8, a

mixed-precision decomposition that made 8-bit inference practical for models with tens of billions of parameters on hardware that would not otherwise hold them in 16-bit precision [11].

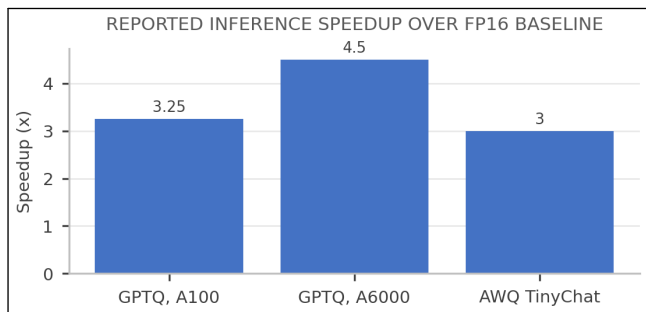


Figure 1: Reported end-to-end inference speedups over an FP16 baseline for three post-training quantization / low-bit inference techniques, as reported by their source papers -- GPTQ on an NVIDIA A100 and on the more cost-effective A6000 [12], and AWQ's TinyChat engine relative to the Hugging Face FP16 implementation [13].

Pushing below 8 bits required more careful handling of quantization error, and two influential post-training quantization methods addressed this from complementary angles. Frantar, Ashkboos, Hoefler, and Alistarh's GPTQ formulates weight quantization as a layer-by-layer reconstruction problem solved with an approximate second-order (Hessian-based) method, and reports that it can quantize a 175-billion-parameter GPT-family model to 3-or-4-bit precision with negligible accuracy degradation in approximately four GPU-hours, for the first time making it possible to run a model of that scale for generative inference inside a single GPU, with reported end-to-end inference speedups over FP16 of around 3.25 times on an NVIDIA A100 and 4.5 times on the more cost-effective NVIDIA A6000 [12]. Lin, Tang, Tang, Yang, Chen, Wang, Xiao, Dang, Gan, and Han's AWQ (activation-aware weight quantization) instead identifies and protects the small fraction — the authors report around 1% — of weight channels that are most "salient" according to the *activation* distribution rather than the weight distribution itself, scaling those channels before quantizing uniformly to 4 bits, and pairs the method with a purpose-built inference engine, TinyChat, that the authors report delivers more than a 3-times speedup over the Hugging Face FP16 implementation on both desktop and mobile GPUs [13]. Xiao et al.'s SmoothQuant addresses a related but distinct obstacle — that *activation* outliers, not just weight outliers, are what make full int8 quantization (weights and activations both) difficult for transformer models — by mathematically migrating quantization

difficulty from activations to weights through a per-channel smoothing factor, enabling accurate 8-bit weight-and-activation quantization without any retraining [14].

At the most aggressive end of the compression spectrum, Dettmers, Pagnoni, Holtzman, and Zettlemoyer's QLoRA demonstrated that a quantized backbone (4-bit NormalFloat quantization, paired with double quantization of the quantization constants themselves and paged optimizer states to avoid memory spikes) combined with a small number of trainable low-rank adapter parameters is sufficient to fine-tune large models cheaply, reporting that the technique reduces memory usage enough to fine-tune a 65-billion-parameter model on a single 48GB GPU while preserving performance comparable to full 16-bit fine-tuning, and that a resulting fine-tuned model family (Guanaco) reached 99.3% of ChatGPT's evaluated performance level after only 24 hours of fine-tuning on a single GPU [15]. QLoRA's technique is aimed primarily at making fine-tuning affordable rather than at the deployed inference target itself, but the underlying 4-bit NormalFloat quantization scheme has been widely adopted directly in edge-deployment pipelines as a way to compress the base model that inference ultimately runs. Hu, Shen, Wallis, Allen-Zhu, Li, Wang, Wang, and Chen's earlier LoRA work, which QLoRA builds directly on, established the low-rank-adapter idea itself — freezing the pretrained weights and injecting small trainable rank-decomposition matrices into each transformer layer — as a parameter-efficient alternative to full fine-tuning, and the same low-rank-adapter mechanism recurs throughout the edge-LLM literature as the standard cheap recovery step after pruning or quantization has degraded a model's accuracy [16].

The most extreme point on the bit-width spectrum abandons multi-bit quantization of a pretrained model altogether in favor of training with an extremely low-precision representation from the start. Ma, Wang, Ma, Wang, Wang, Huang, Dong, Wang, Xue, and Wei's BitNet b1.58 trains transformer weights that take only three possible values — $\{-1, 0, 1\}$, a ternary representation requiring an average of 1.58 bits per weight — and reports that models trained this way match the perplexity and downstream task performance of full-precision (FP16) LLaMA-architecture models at equivalent parameter counts starting from a

3-billion-parameter scale, while offering substantially reduced latency, memory, throughput, and energy consumption, because ternary weights eliminate the need for the multiplication operations that dominate the cost of standard matrix multiplication [17]. If this result holds at the scales and hardware configurations that would matter for microcontroller-class deployment — a question the paper itself evaluates primarily on GPU hardware rather than embedded targets — it represents a genuinely different design point from post-hoc quantization of an already-trained model: precision-as-architecture rather than precision-as-compression-step.

5. Small and Edge-Native Language Models

A parallel strand of work sidesteps the "compress a big model" framing entirely and instead asks what a language model architecture should look like when it is designed, from the first training run, for a sub-billion- or low-single-billion-parameter budget rather than adapted down from a larger design after the fact. Liu et al.'s MobileLLM is explicit about this distinction, arguing that for models with fewer than a billion parameters, architectural choices — depth versus width, embedding sharing between input and output layers, and grouped-query attention (which reduces the memory bandwidth cost of the attention mechanism's key-value cache) — matter more than the conventional wisdom, calibrated on much larger models, that simply scaling data and parameters dominates architectural decisions; the paper reports that its resulting 125M and 350M parameter models achieve accuracy improvements of 2.7 and 4.3 percentage points respectively over prior state-of-the-art models at the same sizes, and that a further layer-sharing variant, MobileLLM-LS, adds another 0.7 and 0.8 points on top of that, with the models additionally reported to approach LLaMA-2 7B's accuracy specifically on API-calling tasks despite the roughly 20 to 56-times smaller parameter count [18].

Zhang, Zeng, Wang, and Lu's TinyLlama takes a complementary approach: rather than redesigning the architecture, keep the standard LLaMA-2 architecture at a 1.1-billion-parameter scale but train it on approximately one trillion tokens across three epochs — far more data relative to model size than the original scaling-law-derived training recipes for models of that size would suggest, following the observation from the broader over-training literature that continuing to train

a small model well past its "compute-optimal" token budget continues to improve its capability, and that this trade — more training compute spent once, in exchange for a smaller, cheaper-to-serve deployed model — is often worthwhile when a model will be served far more times than it is trained [19]. Abdin et al.'s Phi-3 technical report describes a family of models explicitly designed and marketed around on-device deployment, with the report's title, "A Highly Capable Language Model Locally on Your Phone," stating the target constraint directly, and the smallest reported configuration achieving performance the authors describe as comparable to much larger models such as GPT-3.5-class systems and Mixtral 8x7B despite its far smaller footprint, an outcome the authors attribute primarily to careful training-data curation and filtering rather than to architectural novelty [20].

These purpose-built small models and the compression techniques discussed in the preceding two sections are not competing strategies so much as complementary stages of the same pipeline in current practice: a small-but-not-tiny purpose-trained base model (a MobileLLM- or TinyLlama-scale checkpoint, say) is then quantized and, if needed, further pruned or distilled down to a still-smaller student before deployment. The architectural lesson from MobileLLM in particular — that design choices calibrated on frontier-scale models do not automatically transfer to the sub-billion-parameter regime — is a useful corrective for anyone treating "shrink a large model" and "design a small model well" as the same problem, and the growing convergence of both research communities on the sub-1B to low-single-digit-billion parameter range as the practical edge-LLM target is itself informative about where the field currently believes the achievable frontier sits for genuinely constrained hardware.

6. Serving Systems and Runtime Engines for Constrained Devices

A compressed model still needs an inference engine capable of actually exploiting its reduced footprint, and a substantial systems literature has grown up around exactly this problem, distinct from — but essential to — the model-compression research discussed above. David et al.'s TensorFlow Lite Micro describes the runtime and tooling layer for the more extreme end of this problem: an open-source inference framework explicitly designed to run on microcontroller-class

embedded devices with severe memory constraints, addressing operational realities — the absence of dynamic memory allocation, the absence of an operating system, extremely limited code and data memory — that do not arise at all when the deployment target is a phone or a laptop-class edge device, and establishing much of the tooling baseline the broader TinyML community has since built on [2]. This paper is a useful reminder that "edge" spans an enormous range of hardware capability, and that the generative-LLM serving systems discussed next are, almost without exception, targeting the considerably more capable end of that range — phone-class systems-on-chip, single-board computers, or consumer GPUs — rather than true microcontroller-class hardware, a gap we return to in the following section.

Within that more capable tier, several systems papers address the specific problem of serving an already-compressed generative LLM efficiently once memory, rather than raw compute, becomes the binding constraint. Alizadeh, Mirzadeh, Belenko, Khatamifard, Cho, Del Mundo, Rastegari, and Farajtabar's "LLM in a Flash" addresses devices where a model's weights do not fit in available DRAM at all, storing parameters in flash storage and loading only the needed subset into DRAM on demand, using a windowing technique that reuses previously activated neurons to reduce redundant data transfer and a row-column bundling technique that groups data to better exploit flash memory's efficient sequential-access pattern; the authors report that these techniques enable running models up to twice the size of a device's available DRAM, with substantial inference-speed improvements over naive on-demand loading [21]. Song, Mi, Xie, and Chen's PowerInfer takes a different systems approach to a related resource-scarcity problem — running large models on a single consumer-grade GPU rather than a datacenter accelerator — by exploiting the empirical observation that neuron activations in LLM inference follow a power-law distribution, so that a comparatively small set of "hot" neurons account for most activations and can be kept resident on the GPU while "cold" neurons are computed on the CPU on demand, and report that this hybrid CPU-GPU approach outperforms llama.cpp, a widely used reference inference engine, by up to 11.69 times on a single NVIDIA RTX 4090 while retaining model accuracy [22]. Kwon et al.'s vLLM addresses the adjacent problem of serving *many* concurrent requests

efficiently rather than a single request as fast as possible, applying an operating-systems-style virtual-memory technique, PagedAttention, to the transformer key-value cache to eliminate the memory fragmentation and over-allocation that naive contiguous KV-cache allocation produces, and reporting 2-to-4-times throughput improvements over comparable serving systems such as FasterTransformer and Orca while maintaining comparable latency [23]. Although vLLM's original evaluation targets datacenter-class serving rather than a single constrained device, the KV-cache memory-efficiency problem it solves recurs, in a smaller and correspondingly more severe form, on any edge device attempting to serve even a modest context length.

Two further systems-level techniques recur across the constrained-inference literature and deserve separate mention because they address the specific arithmetic and memory-access patterns of transformer inference rather than compressing the model itself. Dao, Fu, Ermon, Rudra, and Ré's FlashAttention restructures the self-attention computation to be aware of the GPU memory hierarchy, using tiling to avoid materializing the full attention matrix in slow high-bandwidth memory, and reports roughly a 3-times speedup on GPT-2 training and inference along with the ability to handle substantially longer context lengths than prior exact-attention implementations, at essentially no accuracy cost since the computation remains mathematically exact rather than approximate [24]. Leviathan, Kalman, and Matias's speculative decoding technique instead accelerates autoregressive generation itself by using a small, cheap draft model to propose several tokens at once, which the large target model then verifies in a single parallel forward pass rather than one sequential step per token, reporting 2-to-3-times inference speedups on a T5-XXL model with results provably identical to standard sequential decoding from the target model [25]. Speculative decoding is a particularly natural fit for the edge-LLM problem specifically because a distilled or pruned small model of exactly the kind surveyed elsewhere in this article is a ready-made draft model, and a growing share of edge-deployment pipelines pair a small on-device model with speculative-decoding-style verification against a larger model when connectivity to one is available, rather than treating the small model purely as a standalone deployment target.

Finally, it is worth noting that architectural alternatives to the standard attention mechanism have also been proposed with efficiency, rather than raw capability, as an explicit design goal. Gu and Dao's Mamba replaces self-attention with a selective state-space mechanism whose computational and memory cost scales linearly, rather than quadratically, with sequence length, and the authors report roughly 5-times higher inference throughput than comparably sized Transformers along with linear scaling in sequence length, a property that is particularly attractive for the streaming, long-running sensor and dialogue workloads common in edge deployments even though the paper's own evaluation, like most of the systems work surveyed here, targets GPU rather than embedded hardware [26]. Miao, Oliaro, Zhang, Cheng, Jin, Chen, and Jia's survey of efficient generative LLM serving usefully organizes this entire systems layer — attention and decoding algorithms, memory management, batching and scheduling, hardware-aware kernel design — into a single taxonomy spanning algorithms through to full serving systems, and is a useful map for a reader trying to understand how the individually narrow techniques surveyed in this section fit together into a deployed serving stack [27].

7. Open Challenges

Several gaps recur across the literature surveyed here and are worth stating plainly. First, and most consequential for the title of this article specifically: there is a substantial and largely unacknowledged gap between "edge" as evaluated in most of the systems papers above — a phone-class system-on-chip, a single consumer GPU, a Raspberry Pi-class single-board computer — and "constrained embedded devices" in the TinyML sense of hundreds of kilobytes to a few megabytes of RAM and milliwatt power budgets, the regime David et al.'s TensorFlow Lite Micro framework was built for [2]. Wan, Wang, Liu, Alam, Zheng, Liu, Qu, Yan, Zhu, Zhang, Chowdhury, and Zhang's survey of efficient large language models, which organizes the field's efficiency techniques from model-, data-, and framework-centric perspectives, is candid that the great majority of reported "edge" and "on-device" LLM results target the more capable end of this spectrum, and true microcontroller-class deployment of anything resembling a modern generative language model — as opposed to small classification or keyword-spotting models — remains largely aspirational rather than demonstrated at scale [28]. Closing this gap will likely

require techniques from several of the sections above composed far more aggressively than any single paper surveyed here has yet demonstrated: BitNet-style ternary weights or comparably extreme quantization, MobileLLM-scale or smaller purpose-built architectures, and TinyML-grade runtime engines, evaluated together on genuinely microcontroller-class hardware rather than any one of these in isolation on a GPU.

Second, evaluation and benchmarking practice across this literature is inconsistent in ways that make cross-paper comparison harder than it should be: reported speedups are measured against different baselines (FP16 on a specific GPU, llama.cpp, a different quantization method entirely), on different hardware, at different batch sizes and context lengths, and accuracy retention is reported on different benchmark suites depending on whether the underlying model is encoder-only, decoder-only, instruction-tuned, or a base pretrained checkpoint. A reader trying to decide, for a specific embedded target, whether GPTQ, AWQ, or a from-scratch small model like MobileLLM is the better starting point has to reconstruct an apples-to-apples comparison largely by hand, because the source papers rarely evaluate against each other directly under matched conditions. Third, the interaction between compression techniques is underexplored relative to how well each technique is understood in isolation: pruning followed by quantization followed by distillation is a plausible and increasingly common pipeline in practice, but few of the papers surveyed here characterize how error compounds or interacts across such a pipeline, as opposed to evaluating each technique applied to an otherwise-unmodified baseline model.

Fourth, energy and thermal behavior — as opposed to raw latency or memory footprint — remains a secondary consideration in most of this literature despite being frequently the binding constraint for a battery-powered embedded deployment specifically. Several papers report throughput and memory numbers in considerable detail while reporting energy consumption only qualitatively or not at all, which is a real gap for a title explicitly about constrained embedded devices, where a persistent inference workload's energy draw, not its raw speed, is very often what determines whether a battery-powered deployment is viable at all. Fifth, and finally, on-device adaptation and personalization for

edge-native language models — updating a deployed small model's behavior in light of local, on-device data without a full retraining cycle — is comparatively underdeveloped relative to the static, train-once-then-deploy pattern that essentially every technique surveyed in this article follows; this stands in some contrast to the more mature literature on on-device adaptation for other model classes, and represents, in our reading, one of the more promising directions for the field to take up next.

8. Conclusion

The literature reviewed here supports a reasonably specific claim: edge-native large language model deployment is not the product of any single technique but a composition of at least four largely separable research threads — knowledge distillation, which trains a smaller model to imitate a larger one's behavior rather than learning a task from scratch [1,4,6]; structured pruning, which removes components directly from an already-trained large model and recovers the resulting accuracy loss cheaply [8,9,10]; quantization, which reduces the bit-width of the remaining weights and activations, at the most aggressive end down to ternary representations trained as such from the start [12,13,17]; and purpose-built small models and runtime engines that are designed, respectively, for a constrained parameter budget and for a constrained memory-and-compute budget from the outset rather than adapted down after the fact [18,19,2]. Read together, these threads paint a coherent and increasingly mature picture of how to get a capable language model onto hardware that was never designed to host one, with reported compression factors, speedups, and accuracy-retention figures that are consistent enough across independently developed techniques to be taken seriously as an engineering roadmap rather than a collection of one-off results.

What the literature does not yet show, and what we think honest readers of this survey should take away as the central open problem, is a demonstrated, end-to-end path from a modern generative language model to genuinely microcontroller-class embedded hardware of the kind TinyML research has long targeted for smaller model classes [2]. Almost every impressive number reviewed in this article — TinyBERT's 7.5-times compression [4], GPTQ's 3-to-4.5-times inference speedup [12], PowerInfer's 11.69-times speedup over

llama.cpp [22], BitNet's ternary-weight parity with full-precision models [17] — was measured on a GPU or a phone-class chip, not on the hundreds-of-kilobytes-of-RAM hardware the phrase "constrained embedded device" most naturally evokes. Closing that remaining gap, and doing so with the same empirical rigor and cross-technique comparison that this article has argued the field still needs more of, is the clearest unfinished business this literature leaves for future work.

9. References

1. Hinton, G., Vinyals, O., Dean, J. (2015) Distilling the Knowledge in a Neural Network. arXiv preprint. DOI: [10.48550/arXiv.1503.02531](https://doi.org/10.48550/arXiv.1503.02531)
2. David, R., Duke, J., Jain, A., Janapa Reddi, V., Jeffries, N., Li, J., Kreeger, N., Nappier, I., Natraj, M., Regev, S., Rhodes, R., Wang, T., Warden, P. (2021) TensorFlow Lite Micro: Embedded Machine Learning on TinyML Systems. Proceedings of Machine Learning and Systems (MLSys) 3. DOI: [10.48550/arXiv.2010.08678](https://doi.org/10.48550/arXiv.2010.08678)
3. Sanh, V., Debut, L., Chaumond, J., Wolf, T. (2019) DistilBERT, a Distilled Version of BERT: Smaller, Faster, Cheaper and Lighter. arXiv preprint. DOI: [10.48550/arXiv.1910.01108](https://doi.org/10.48550/arXiv.1910.01108)
4. Jiao, X., Yin, Y., Shang, L., Jiang, X., Chen, X., Li, L., Wang, F., Liu, Q. (2020) TinyBERT: Distilling BERT for Natural Language Understanding. Findings of the Association for Computational Linguistics: EMNLP 2020. DOI: [10.48550/arXiv.1909.10351](https://doi.org/10.48550/arXiv.1909.10351)
5. Wang, W., Wei, F., Dong, L., Bao, H., Yang, N., Zhou, M. (2020) MiniLM: Deep Self-Attention Distillation for Task-Agnostic Compression of Pre-Trained Transformers. Advances in Neural Information Processing Systems (NeurIPS) 33. DOI: [10.48550/arXiv.2002.10957](https://doi.org/10.48550/arXiv.2002.10957)
6. Gu, Y., Dong, L., Wei, F., Huang, M. (2023) MiniLLM: Knowledge Distillation of Large Language Models. arXiv preprint. DOI: [10.48550/arXiv.2306.08543](https://doi.org/10.48550/arXiv.2306.08543)
7. Hsieh, C.Y., Li, C.L., Yeh, C.K., Nakhost, H., Fujii, Y., Ratner, A., Krishna, R., Lee, C.Y., Pfister, T. (2023) Distilling Step-by-Step! Outperforming Larger Language Models with Less Training Data and Smaller Model Sizes. Findings of the Association for Computational Linguistics: ACL 2023. DOI: [10.48550/arXiv.2305.02301](https://doi.org/10.48550/arXiv.2305.02301)
8. Ma, X., Fang, G., Wang, X. (2023) LLM-Pruner: On the Structural Pruning of Large Language Models. Advances in Neural Information Processing Systems (NeurIPS) 36. DOI: [10.48550/arXiv.2305.11627](https://doi.org/10.48550/arXiv.2305.11627)
9. Sun, M., Liu, Z., Bair, A., Kolter, J.Z. (2023) A Simple and

- Effective Pruning Approach for Large Language Models. arXiv preprint. DOI: [10.48550/arXiv.2306.11695](https://doi.org/10.48550/arXiv.2306.11695)
10. Xia, M., Gao, T., Zeng, Z., Chen, D. (2023) Sheared LLaMA: Accelerating Language Model Pre-training via Structured Pruning. arXiv preprint. DOI: [10.48550/arXiv.2310.06694](https://doi.org/10.48550/arXiv.2310.06694)
 11. Dettmers, T., Lewis, M., Belkada, Y., Zettlemoyer, L. (2022) LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale. arXiv preprint. DOI: [10.48550/arXiv.2208.07339](https://doi.org/10.48550/arXiv.2208.07339)
 12. Frantar, E., Ashkboos, S., Hoefler, T., Alistarh, D. (2023) GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers. International Conference on Learning Representations (ICLR). DOI: [10.48550/arXiv.2210.17323](https://doi.org/10.48550/arXiv.2210.17323)
 13. Lin, J., Tang, J., Tang, H., Yang, S., Chen, W.M., Wang, W.C., Xiao, G., Dang, X., Gan, C., Han, S. (2023) AWQ: Activation-Aware Weight Quantization for LLM Compression and Acceleration. arXiv preprint. DOI: [10.48550/arXiv.2306.00978](https://doi.org/10.48550/arXiv.2306.00978)
 14. Xiao, G., Lin, J., Seznec, M., Wu, H., Demouth, J., Han, S. (2023) SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models. Proceedings of the 40th International Conference on Machine Learning (ICML). DOI: [10.48550/arXiv.2211.10438](https://doi.org/10.48550/arXiv.2211.10438)
 15. Dettmers, T., Pagnoni, A., Holtzman, A., Zettlemoyer, L. (2023) QLoRA: Efficient Finetuning of Quantized LLMs. Advances in Neural Information Processing Systems (NeurIPS) 36. DOI: [10.48550/arXiv.2305.14314](https://doi.org/10.48550/arXiv.2305.14314)
 16. Hu, E.J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., Chen, W. (2021) LoRA: Low-Rank Adaptation of Large Language Models. arXiv preprint. DOI: [10.48550/arXiv.2106.09685](https://doi.org/10.48550/arXiv.2106.09685)
 17. Ma, S., Wang, H., Ma, L., Wang, L., Wang, W., Huang, S., Dong, L., Wang, R., Xue, J., Wei, F. (2024) The Era of 1-bit LLMs: All Large Language Models are in 1.58 Bits. arXiv preprint. DOI: [10.48550/arXiv.2402.17764](https://doi.org/10.48550/arXiv.2402.17764)
 18. Liu, Z., Zhao, C., Iandola, F., Lai, C., Tian, Y., Fedorov, I., Xiong, Y., Chang, E., Shi, Y., Krishnamoorthi, R., Lai, L., Chandra, V. (2024) MobileLLM: Optimizing Sub-billion Parameter Language Models for On-Device Use Cases. arXiv preprint. DOI: [10.48550/arXiv.2402.14905](https://doi.org/10.48550/arXiv.2402.14905)
 19. Zhang, P., Zeng, G., Wang, T., Lu, W. (2024) TinyLlama: An Open-Source Small Language Model. arXiv preprint. DOI: [10.48550/arXiv.2401.02385](https://doi.org/10.48550/arXiv.2401.02385)
 20. Abdin, M., et al. (2024) Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone. arXiv preprint. DOI: [10.48550/arXiv.2404.14219](https://doi.org/10.48550/arXiv.2404.14219)
 21. Alizadeh, K., Mirzadeh, I., Belenko, D., Khatamifard, K., Cho, M., Del Mundo, C.C., Rastegari, M., Farajtabar, M. (2023) LLM in a Flash: Efficient Large Language Model Inference with Limited Memory. arXiv preprint. DOI: [10.48550/arXiv.2312.11514](https://doi.org/10.48550/arXiv.2312.11514)
 22. Song, Y., Mi, Z., Xie, H., Chen, H. (2023) PowerInfer: Fast Large Language Model Serving with a Consumer-grade GPU. arXiv preprint. DOI: [10.48550/arXiv.2312.12456](https://doi.org/10.48550/arXiv.2312.12456)
 23. Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C.H., Gonzalez, J.E., Zhang, H., Stoica, I. (2023) Efficient Memory Management for Large Language Model Serving with PagedAttention. Proceedings of the 29th ACM Symposium on Operating Systems Principles (SOSP). DOI: [10.48550/arXiv.2309.06180](https://doi.org/10.48550/arXiv.2309.06180)
 24. Dao, T., Fu, D.Y., Ermon, S., Rudra, A., Ré, C. (2022) FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. Advances in Neural Information Processing Systems (NeurIPS) 35. DOI: [10.48550/arXiv.2205.14135](https://doi.org/10.48550/arXiv.2205.14135)
 25. Leviathan, Y., Kalman, M., Matias, Y. (2023) Fast Inference from Transformers via Speculative Decoding. Proceedings of the 40th International Conference on Machine Learning (ICML). DOI: [10.48550/arXiv.2211.17192](https://doi.org/10.48550/arXiv.2211.17192)
 26. Gu, A., Dao, T. (2023) Mamba: Linear-Time Sequence Modeling with Selective State Spaces. arXiv preprint. DOI: [10.48550/arXiv.2312.00752](https://doi.org/10.48550/arXiv.2312.00752)

27. Miao, X., Oliaro, G., Zhang, Z., Cheng, X., Jin, H., Chen, T., Jia, Z. (2023) Towards Efficient Generative Large Language Model Serving: A Survey from Algorithms to Systems. arXiv preprint. DOI: [10.48550/arXiv.2312.15234](https://doi.org/10.48550/arXiv.2312.15234)
28. Wan, Z., Wang, X., Liu, C., Alam, S., Zheng, Y., Liu, J., Qu, Z., Yan, S., Zhu, Y., Zhang, Q., Chowdhury, M., Zhang, M. (2023) Efficient Large Language Models: A Survey. arXiv preprint. DOI: [10.48550/arXiv.2312.03863](https://doi.org/10.48550/arXiv.2312.03863)

10. Declarations

Funding

This literature synthesis received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

Conflict of Interest

The authors declare that they have no conflict of interest.

Data Availability Statement

This article is a literature synthesis and did not generate new datasets or code; all figures and numerical claims are drawn from, and attributed to, the cited sources, which readers should consult for original data and code availability.

Ethical Approval

Not applicable. This work did not involve human participants, animal subjects, or the collection of primary experimental data.