

LANGUAGE AND TRANSFORMER MODELS WITH ESP32-CLASS MICROCONTROLLERS: A SYSTEMATIC REVIEW OF HYBRID DEPLOYMENT AND ON-DEVICE OPTIMIZATION

Muhammad Rasyid Ramdhani^{1)*}, Darmawan Hidayat¹⁾ and Agus Trisanto¹⁾

¹⁾Departemen Teknik Elektro, Universitas Padjadjaran, Jatinangor, Sumedang 45363, Indonesia

(Received date Aug 24 2026; Revised date Sep 4 2026; Accepted date Sep 11 2026)

ABSTRACT—Deploying language and transformer models with ultra-low-power microcontrollers creates a fundamental tension between model capability and the memory, energy, and processing limits of embedded hardware. This systematic review examines how ESP32-class devices participate in language-model systems and when transformer inference can be executed directly on the microcontroller. Following a PRISMA-guided search completed in June 2026, ten studies were included. To avoid conflating an LLM-enabled ESP32 system with LLM inference on the ESP32 itself, the evidence is organized into three deployment classes: cloud-assisted systems, local-host/edge-assisted systems, and fully microcontroller-resident transformer or small-language-model inference. The first two classes preserve access to larger models but introduce host or network dependencies. Fully on-device studies rely on aggressive compression and task specialization: reported language/foundation-model implementations operate within approximately 256–412 KB of RAM, while a fully ternarized Vision Transformer demonstrates a 5.81 MB model on an ESP32-S3 configuration with external PSRAM. Across the evidence base, model-size reductions of up to 63× are reported, but accuracy, latency, and energy values are not directly comparable because tasks and evaluation protocols differ substantially. No included study demonstrates a general-purpose conversational LLM running entirely on an ESP32. The review therefore identifies standardized benchmarking, transparent reporting of inference location and memory hierarchy, scalable compression, and hardware-software co-design as priorities for future research.

KEY WORDS : Language Models, Transformers, Edge AI, ESP32, TinyML, Model Compression

1. INTRODUCTION

The proliferation of the Internet of Things (IoT) and edge computing has created strong demand for intelligent, autonomous devices capable of interpreting natural-language or other high-level inputs near the point of data generation. ESP32-family microcontrollers are widely used in this ecosystem because they combine low cost, wireless connectivity, embedded peripherals, and mature development tooling. The original ESP32 provides a dual-core Xtensa LX6 processor operating at up to 240 MHz with approximately 520 KB of SRAM, while the ESP32-S3 uses dual-core Xtensa LX7 cores at up to 240 MHz with 512 KB of internal SRAM and support for external PSRAM. Both platforms support hardware-accelerated single-precision floating-point operations, and the ESP32-S3 additionally provides vector instructions intended to accelerate neural-network and signal-processing workloads (Espressif Systems, 2026a; Espressif Systems, 2026b). These capabilities are substantial for a microcontroller, but remain several orders of magnitude below the memory and compute resources normally associated with contemporary general-purpose language models.

This resource gap makes terminology especially important. In this review, a large language model (LLM) denotes a large generative language model whose full inference workload would normally require substantially more memory than an ESP32-class microcontroller can provide; a small language model (SLM) denotes a compact language model designed for constrained inference; and a transformer model includes both language transformers and non-language architectures such as Vision Transformers (ViTs). We also distinguish an LLM-enabled embedded system, in which an ESP32 participates as a sensor, interface, communication endpoint, or actuator while the LLM executes on a cloud service or local host, from fully on-device inference, in which the transformer computation itself is executed on the microcontroller. This distinction is essential because an ESP32 connected to ChatGPT, Gemini, DeepSeek, or another remote model is architecturally different from an ESP32 executing a compressed transformer locally.

Despite growing interest in TinyML and edge AI, the evidence relevant to transformer and language-model use with ESP32-class hardware remains fragmented. Existing studies address different layers of the problem: some focus on cloud or host integration, others on safe host-to-microcontroller control, and a smaller subset investigates model compression, quantization, or tensor decomposition for true microcontroller-resident inference. These studies also report heterogeneous outcomes, including command-recognition accuracy, application-level response time, model compression ratio, memory footprint, ImageNet accuracy, or collision rate. Consequently, raw numerical values

cannot be interpreted as if they were measured on a common task or benchmark. A systematic synthesis therefore requires both a deployment taxonomy and explicit separation between within-study results and cross-study comparison.

The objective of this systematic review is to provide a structured and critical synthesis of deployment strategies that bring language- and transformer-based intelligence to ESP32-class microcontrollers. The main contributions are fourfold. First, we define an inference-location taxonomy that separates cloud-assisted, local-host/edge-assisted, and fully microcontroller-resident deployments. Second, we synthesize the model-level techniques used for on-device feasibility, including tensor decomposition, ternary quantization, knowledge distillation, and quantization-aware recovery. Third, we compare reported resource and performance metrics without treating heterogeneous task metrics as directly equivalent. Finally, we identify methodological and engineering gaps, particularly the absence of a standardized benchmark for MCU-resident language inference and the lack of evidence for a general-purpose conversational LLM executing entirely on an ESP32.

The remainder of this paper is organized as follows. Section 2 describes the review protocol, search strategy, operational definitions, eligibility criteria, study selection, and quality appraisal. Section 3 reports publication trends. Section 4 summarizes the included studies and synthesizes assisted deployment architectures and fully on-device optimization techniques. Section 5 discusses the implications, limitations, and future research directions. Section 6 concludes the review.

2. METHODOLOGY

This systematic review was conducted using the Preferred Reporting Items for Systematic Reviews and Meta-Analyses (PRISMA 2020) as the reporting framework (Page et al., 2021). The review process was designed to make the literature search, screening logic, evidence classification, and data extraction transparent. Because the topic spans embedded systems, machine learning, and system integration, the review includes both model-level studies that execute transformer inference on microcontrollers and system-level studies in which ESP32-class devices form part of an LLM-enabled architecture. The distinction between these evidence types is maintained throughout the analysis rather than treating all studies as examples of LLM inference on the microcontroller itself.

2.1. Review Protocol

A review protocol was defined before the final literature search to reduce selection bias and ensure consistent screening. The overarching question was formulated using the SPICE framework: Setting (ESP32 and closely related resource-constrained microcontrollers),

Perspective (deployment and optimization), Intervention (compression, quantization, system partitioning, and assisted architectures), Comparison (alternative inference locations or uncompressed/full-precision baselines where available), and Evaluation (accuracy or task metric, latency, memory, model footprint, energy/power, and system reliability). Three operational research questions guided the synthesis: RQ1, where is model inference executed and what role does the ESP32 perform; RQ2, which model-level techniques enable transformer or language-model inference on microcontroller-class hardware; and RQ3, what resource-performance trade-offs are reported and which comparisons are methodologically valid. Five primary academic sources—IEEE Xplore, ACM Digital Library, SpringerLink, arXiv, and Scopus—were searched, with Google Scholar used as a supplementary source to capture additional interdisciplinary and grey literature. Extracted fields included publication year, model class, inference location, microcontroller platform, task, optimization technique, model and memory footprint, latency, accuracy or task-specific metric, energy/power where available, and reproducibility information.

The search strategy combined three conceptual blocks: target hardware, model type, and deployment context. The core expression used for IEEE Xplore and ACM Digital Library was ("tinyML" OR "microcontroller" OR "ESP32") AND ("large language model" OR "LLM" OR "small language model" OR "transformer" OR "neural network") AND ("on-device" OR "edge AI" OR "embedded" OR "inference"). Equivalent terms and field syntax were adapted for Scopus, SpringerLink, and arXiv, while Google Scholar was queried with simplified combinations such as ESP32 LLM inference and microcontroller transformer deployment. The final search was conducted in June 2026 and was limited to English-language studies. No lower publication-year bound was imposed because the objective was to capture the complete development of this emerging topic up to the June 2026 search cutoff.

2.2. Scope, Operational Definitions, and Thematic Taxonomy

To avoid conflating system integration with true on-device model execution, the included studies were classified primarily by inference location. Class A comprises cloud-assisted LLM systems in which the ESP32 performs sensing, communication, user-interface, or actuation functions while the language model executes on an external cloud service. Class B comprises local-host or edge-assisted systems in which the LLM executes on a nearby computer, home server, Raspberry Pi-class host, or other companion device and the ESP32 performs deterministic low-level execution or state feedback. Class C comprises fully microcontroller-resident transformer or language-model inference in which the model computation itself runs on

the ESP32 or a closely related MCU. Non-language transformer studies, such as Vision Transformer deployment, were included only when they provided direct empirical evidence about transformer compression, memory hierarchy, integer-oriented kernels, or hardware feasibility on ESP32-class devices. This taxonomy allows architectural evidence to inform the review without incorrectly labeling every LLM-enabled ESP32 system as an on-device LLM implementation.

2.3. Inclusion and Exclusion Criteria

Studies were included when they met all of the following criteria: (1) the work addressed an LLM, SLM, transformer model, or LLM-enabled embedded architecture involving an ESP32 microcontroller or a closely related resource-constrained MCU; (2) it presented empirical results, a working system implementation, or sufficiently detailed architectural modifications to identify the model-inference location and the role of the microcontroller; (3) it was published as a peer-reviewed journal article, conference paper, openly available thesis, technical report, or preprint with enough technical detail for extraction; (4) it was written in English; and (5) the full text was accessible to the authors. For non-language transformers, an additional requirement was direct empirical deployment on ESP32-class hardware or equivalent evidence that materially informs transformer execution under microcontroller memory and compute constraints.

Studies were excluded when they met any of the following criteria: (1) they discussed general edge AI or TinyML without an explicit language-model or transformer component; (2) they evaluated compression or inference only on a desktop GPU or general-purpose computer without an ESP32-class implementation or a clearly defined ESP32 role; (3) they described AI-enabled IoT at a conceptual level without sufficient implementation detail to determine where inference occurs; (4) they were opinion pieces, blog posts, or non-archival material without technical depth; (5) they duplicated results published elsewhere, in which case the most complete version was retained; or (6) the full text could not be retrieved. These criteria were applied consistently to both system-integration and model-optimization studies.

2.4. Study Selection Process

The study selection process was conducted in four stages following the PRISMA flow diagram shown in Figure 1. The database and supplementary searches yielded 588 records. After 270 duplicates were removed, 318 unique records remained for title-and-abstract screening. Two reviewers independently screened these records against the predefined eligibility criteria, resolving disagreements through discussion and consulting a third reviewer when consensus could not be

reached. This stage excluded 282 records and left 36 reports for full-text retrieval. Seven reports could not be retrieved, leaving 29 reports for full-text eligibility assessment. Nineteen reports were then excluded because they lacked empirical ESP32-class evidence ($n=8$), focused on general TinyML without a relevant transformer or language-model component ($n=6$), provided insufficient technical detail ($n=3$), or duplicated another publication ($n=2$). The final evidence base comprised 10 studies.

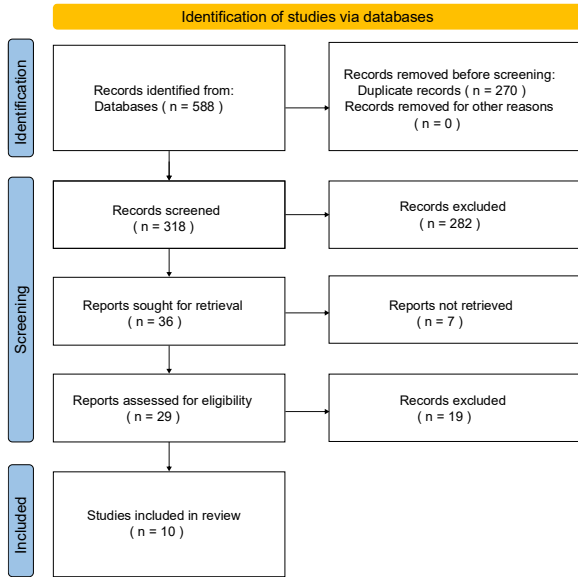


Figure 1. PRISMA flow diagram illustrating the study selection process from initial database search to final inclusion.

A structured quality appraisal was applied to all included studies using four dimensions: Q1, clarity of the research objective; Q2, adequacy of the methodology, including the hardware, model architecture, and deployment or compression procedure; Q3, completeness of reported outcomes, including relevant performance and resource metrics; and Q4, reproducibility, assessed from code/model availability or sufficiently detailed implementation descriptions. Each dimension was scored from 0 (not addressed) to 2 (fully addressed), yielding a maximum score of 8. Total scores of 6–8 were categorized as high quality, 4–5 as moderate quality, and below 4 as low quality. Table 1b reports the Q1/Q2/Q3/Q4 score vector, total score, and category for every included study. Five studies were categorized as high quality and five as moderate quality; none scored below the inclusion threshold. The appraisal is used as evidence-quality context rather than as a numerical ranking across heterogeneous system and algorithm studies.

Several limitations of the review process should be acknowledged. First, restricting the search to English-language publications may exclude relevant work reported in other languages. Second, although the

selected databases cover the main engineering and computing literature, relevant studies indexed only in regional or institutional repositories may have been missed; Google Scholar was therefore used only as a supplementary source because its search results are less reproducible than bibliographic databases. Third, the inclusion of preprints, theses, and conference papers reflects the rapid pace and small size of this research area but also creates variation in peer-review status and reporting depth. Fourth, the search cutoff was June 2026, so studies appearing after that date are outside the review. Finally, the evidence base contains only ten studies and is highly heterogeneous; the results are therefore synthesized narratively rather than pooled in a meta-analysis.

3. RESULTS

3.1. Research Trends

The ten included studies span the period from 2023 to 2026. Based on the bibliographic year of the versions included in this review, one study was published in 2023, one in 2024, three in 2025, and five in 2026, as shown in Figure 2. The increase in publications indicates growing activity in this niche, but the absolute evidence base remains too small to infer a stable long-term publication trend.

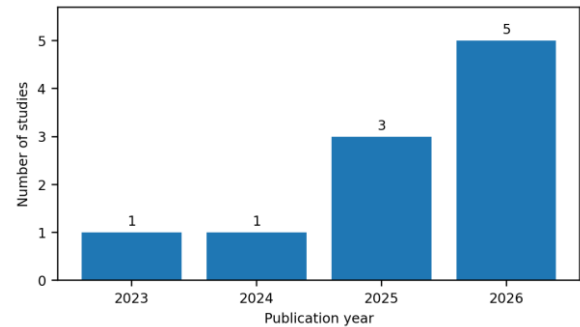


Figure 2. Publication-year distribution of the 10 included studies.

The chronological pattern also reflects a change in the type of evidence being reported. The 2023 study primarily demonstrated server-backed ChatGPT integration with an ESP32 endpoint (Šćulac, 2023). The 2024 study examined compact on-device language-model optimization (Pujari et al., 2024). Studies published in 2025 largely emphasized LLM-enabled embedded applications and cloud or host integration, including ESP32/ESP8266 comparisons, DeepSeek voice assistance, and a handwriting-controlled IoT system (Gupta and Gupta, 2025; Ouyang et al., 2025; Chirath, 2025). By 2026, the literature included both assisted architectures and stronger evidence of microcontroller-resident transformer execution, including MPS-based compression, a fully ternarized Vision Transformer on ESP32-S3, local-host safety

Ramdhani et al.

shielding, and smart-home systems (Elumalai and Rajan, 2026; Ruciński et al., 2026; Xiong and Bao, 2026; Katuwal and Ghimire, 2026; Pikus et al., 2026). This progression supports analyzing the field by inference location rather than by publication count alone.

4. OVERVIEW OF INCLUDED STUDIES

Table 1a summarizes the technical characteristics of the included studies, including model or system, microcontroller platform, inference location, task and key contribution, and reported metrics. Table 1b reports the study-level quality-appraisal scores.

The evidence base is intentionally heterogeneous because it addresses two related but distinct engineering questions: how an ESP32 can participate in an LLM-

enabled system, and how transformer inference can be compressed sufficiently to run on a microcontroller. Accordingly, the synthesis emphasizes architectural classification and within-study trade-offs. Accuracy values from predictive maintenance, image classification, speech recognition, smart-home control, and robotic navigation are not treated as directly comparable measures of model quality.

Table 1a. Characteristics of the included studies, classified by model type and inference location.

STUDY	MODEL / SYSTEM	MCU	INFERENCE LOCATION	TASK / KEY CONTRIBUTION	REPORTED METRICS
Chirath (2025)	Gemini 2.0 Flash API	ESP32	Cloud/host-assisted; external LLM on laptop	Handwriting IoT control; multimodal interpretation + ESP32 actuation	>95% reliability; 3.27 s model-processing; 54.4 s total latency
Elumalai & Rajan (2026)	MPS-compressed foundation models	STM32 H743; ESP32-S3; GD32VF103	Fully MCU-resident after post-training compression	Predictive maintenance; MPS decomposition + LSBA	44–63× compression; 93.8–96.4% task accuracy; 19–142 ms on ESP32-S3; 1.4–4.7 mJ; 389–412 KB peak SRAM
Gupta & Gupta (2025)	ChatGPT; Gemini	ESP8266	Cloud-assisted / external-model integration	AI/NLP applications; ESP32/ESP8266 integration comparison	Primarily qualitative evidence; no common ESP32-resident LLM benchmark
Katuwal & Ghimire (2026)	Groq LLM + Deepgram STT/TTS	S3	Cloud-assisted STT-LLM-TTS pipeline	Smart-home voice control; context-aware LLM + relay actuation	1.2 s voice-to-relay response; >96% command recognition in normal ambient noise
Ouyang et al. (2025)	DeepSeek LLM	ESP32	LLM-assisted; exact inference location not established	Voice assistant; DeepSeek integration with ESP32 platform	Improved response speed and speech recognition reported; limited comparable resource metrics
Pikus et al. (2026)	Polka 1.1B; Bielik 11Bv2	Boneio	Local-host assisted; model runs with Home Assistant/host	Smart-home control; >10k-prompt dataset; fine-tuning/LoRA; ESP32 execution layer	Real-time smart-home execution reported; no claim of ESP32-resident LLM inference
Pujari et al. (2024)	Small language models	M7; ESP32	Fully MCU-resident optimized inference	SLM inference; sparse ternary quantization + federated fine-tuning	92% of full-precision baseline accuracy; 256 KB RAM; 4.3× latency reduction; 5.8× model-size reduction
Ruciński et al. (2026)	FTerViT / DeiT-III-Small	S3-EYE	Fully MCU-resident transformer inference with external PSRAM	Image classification; fully ternary ViT + distillation/QAT recovery	79.64% ImageNet-1K top-1; 5.81 MB flash; 2.83 MB peak PSRAM; 21.06 s full forward pass
Xiong & Bao (2026)	Locally deployed host-side LLM	S3	Local-host assisted; LLM on host, ESP32 deterministic control	Natural-language device control; state-aware prompt + runtime shield	~0.15 s host decision; 30.6%/26.5% unshielded collisions; 0% for shielded variants in evaluated scenarios
Ščulac (2023)	ChatGPT	ESP32	Cloud/server-assisted through Django	Voice assistant; server-backed ChatGPT + ESP32 microphone/speaker	Proof-of-concept; no directly comparable MCU-resident language-inference metric

Table 1b. Study-level quality appraisal. Q1 = objective clarity; Q2 = methodological adequacy; Q3 = results completeness; Q4 = reproducibility; H = high; M = moderate.

STUDY	Q1	Q2	Q3	Q4	TOTAL	CATEGORY
Chirath (2025)	2	1	1	0	4	M
Elumalai & Rajan (2026)	2	2	2	1	7	H
Gupta & Gupta (2025)	2	1	1	0	4	M
Katuwal & Ghimire (2026)	2	1	1	0	4	M
Ouyang et al. (2025)	2	1	1	0	4	M
Pikus et al. (2026)	2	2	2	1	7	H
Pujari et al. (2024)	2	2	2	1	7	H
Ruciński et al. (2026)	2	2	2	2	8	H
Xiong & Bao (2026)	2	2	2	1	7	H
Ščulac (2023)	2	1	1	0	4	M

4.1. System Integration and Assisted Deployment Architectures

Seven of the ten included studies use the ESP32 primarily as part of a larger system rather than as the processor executing the full language model. These architectures are important because they show how LLM capability can be integrated with low-cost embedded hardware, but they must be distinguished from microcontroller-resident inference. Table 2 therefore classifies system-level studies by the physical location of LLM inference and the role assigned to the ESP32.

Cloud-assisted pipelines are the most common system-level pattern. Katuwal and Ghimire (2026) use an ESP32-S3 with a MEMS microphone, but speech recognition, language understanding, and speech synthesis are delegated to Deepgram and Groq services. The reported 1.2 s voice-command-to-relay response and >96% command-recognition accuracy are therefore application-level results for a networked pipeline, not evidence of LLM inference on the ESP32. Chirath (2025) similarly combines a webcam, laptop, Gemini 2.0 Flash API, and ESP32 actuator controller. Although the study describes an edge-deployable and offline-capable concept, its reported implementation invokes an external multimodal LLM API; accordingly, this review classifies it as cloud-/host-assisted. The reported >95% command reliability, 3.27 s model-processing time, and 54.4 s total system latency should be interpreted in that system context.

A distinct pattern is local-host assistance, where cloud dependency is removed but the LLM still does not execute on the microcontroller. EdgeTalk-MCU (Xiong and Bao, 2026) couples a locally deployed LLM on a host computer with an ESP32-S3 through a lightweight serial protocol. The host generates high-level action proposals, while the microcontroller executes structured commands and returns state feedback. The reported ~0.15 s decision latency refers to the host-side inference pipeline, with additional UART and actuator delay on the physical platform. In the reported Sudden Obstacle

scenario, unshielded LLM-only and prompt-only controllers produced collision rates of 30.6% and 26.5%, respectively, whereas the evaluated shielded variants produced 0% collisions. This is evidence for the effectiveness of the runtime shield under the reported scenarios, not a universal safety guarantee.

Pikus et al. (2026) use Home Assistant as the central automation platform and Polka 1.1B or Bielik 11Bv2 for natural-language interpretation, with ESP32-based Boneio controllers forming the physical execution layer. Polka is fine-tuned using a conventional process, whereas Bielik is adapted with Low-Rank Adaptation (LoRA); a custom prompt generator is used to create more than 10,000 training examples. The key contribution for the present review is architectural: the relatively large language models execute on a more capable host, while the ESP32 handles deterministic actuation. The architecture therefore demonstrates practical edge-local LLM integration, not MCU-resident LLM inference.

The remaining assisted studies provide additional system-integration evidence. Šćulac (2023) uses a Django server for speech processing, user management, and ChatGPT access while the ESP32 provides a microphone/speaker endpoint. Gupta and Gupta (2025) compare ESP32 and ESP8266 in AI-powered applications and discuss integration with ChatGPT and Gemini, but do not provide a common on-device LLM benchmark. Ouyang et al. (2025) describe a DeepSeek-based intelligent voice assistant on an ESP32 platform and report improved response and speech-recognition performance, but the available implementation description is insufficient to establish that DeepSeek inference occurs on the microcontroller itself. These studies are therefore retained as assisted-deployment evidence but are not counted as demonstrations of an LLM running on ESP32.

Table 2 summarizes the inference location, ESP32 role, and principal system-level evidence for the assisted deployment studies.

Table 2. Assisted deployment architectures classified by inference location.

STUDY	CLASS	INFERENCE LOCATION / ESP32 ROLE	SYSTEM EVIDENCE
Chirath (2025)	assisted	Cloud/host- Gemini API runs from laptop; ESP32 controls relays/servo	>95% reliability; 3.27 s model; 54.4 s total; external multimodal LLM API
Šćulac (2023)	r-assisted	Cloud/serve ChatGPT and speech processing on Django server; ESP32 microphone/speaker endpoint	Proof-of-concept; network/server dependency; no MCU-resident language-inference metric
Gupta & Gupta (2025)	assisted / comparative	Cloud- External ChatGPT/Gemini services; ESP32/ESP8266 used as embedded endpoints	Qualitative hardware/integration comparison; no common ESP32 on-device benchmark
Katuwal & Ghimire (2026)	assisted	Cloud- Deepgram STT → Groq LLM → Deepgram TTS; ESP32-S3 handles audio and relays	1.2 s voice-to-relay; >96% command recognition under normal ambient noise
Ouyang et al. (2025)	assisted; location unspecified	LLM- ESP32 voice-assistant platform combined with DeepSeek; available report does not establish MCU-resident LLM inference	Response-speed and speech-recognition improvement reported; limited resource details
Pikus et al. (2026)	assisted	Local-host Polka/Bielik execute with Home Assistant/host; ESP32 Boneio controllers execute device actions	Real-time smart-home architecture; host-side language model, not ESP32-resident inference
Xiong & Bao (2026)	assisted	Local-host Local LLM on host; ESP32-S3 executes deterministic commands and returns state feedback	~0.15 s host decision; 0% shielded collisions in evaluated scenarios vs 30.6%/26.5% unshielded; HIL validation

The central finding from the assisted architectures is that system-level success does not imply that the

language model executes on the ESP32. Cloud-assisted systems can use very capable models but inherit

network availability, service latency, privacy, and external-service dependencies. Local-host systems remove the cloud requirement and can provide low-latency, privacy-preserving operation within a local network, but still require a companion computer with substantially greater memory and compute resources than the microcontroller. For practitioners, inference location should therefore be reported as a first-class system metric alongside accuracy and latency.

4.2. Fully On-Device Model Compression and Transformer Deployment

True microcontroller-resident transformer inference is constrained primarily by memory capacity, memory bandwidth, and the cost of repeated matrix operations. The original ESP32 has approximately 520 KB of SRAM, whereas the ESP32-S3 provides 512 KB of internal SRAM and can access external PSRAM depending on the module configuration. Contrary to a common simplification, both the ESP32 and ESP32-S3 provide hardware support for single-precision floating point; however, integer and low-bit arithmetic remain important because they reduce model storage, memory traffic, and computational cost. The ESP32-S3 also provides vector instructions that can accelerate neural-network and signal-processing workloads (Espressif Systems, 2026a; Espressif Systems, 2026b). The included on-device studies therefore rely on aggressive compression rather than on the absence of floating-point hardware.

The MPS-TinyFM framework (Elumalai and Rajan, 2026) reports the most aggressive model-size reduction among the language/foundation-model studies. The method reshapes trained weight matrices into higher-order tensors and factorizes them using Matrix Product State (MPS) decomposition. Layer-Sensitivity Bond Dimension Allocation (LSBA) assigns different bond dimensions according to layer sensitivity, while inference proceeds directly on the factor chain rather than reconstructing the original weights. Across the evaluated foundation-model and MCU configurations, the authors report 44–63 $\times$ compression and 93.8–96.4% task-specific accuracy, with ESP32-S3 inference latency of approximately 19–142 ms and peak runtime SRAM of 389–412 KB. These results show that substantial compression can make specialized foundation-model workloads feasible within MCU-scale memory, but the metrics are task-specific and should not be interpreted as equivalent to conversational LLM performance.

Pujari et al. (2024) investigate compact small-language-model deployment using sparse ternary quantization and federated fine-tuning. Ternary weights take values in $\{-1, 0, +1\}$, and sparsity reduces both storage and matrix-operation cost. The study reports operation within 256 KB of RAM, a 5.8 $\times$ model-size reduction, a 4.3 $\times$ latency reduction relative to the study baseline, and retention of approximately 92% of full-

precision baseline accuracy. The work also incorporates privacy-oriented differential-noise injection during preprocessing. Its main relevance is that it targets a much tighter memory budget than conventional edge inference, although the reported task metrics remain specific to the evaluated SLM workloads.

FTerViT (Ruciński et al., 2026) provides direct evidence that a substantially larger transformer architecture can execute fully on an ESP32-S3 when the memory hierarchy includes external PSRAM and the model is aggressively quantized. The method ternarizes all weight matrices and normalization parameters and introduces TernaryBitConv2d and TernaryLayerNorm, trained through knowledge distillation followed by a lightweight quantization-aware recovery phase. The 384×384 model achieves 82.43% ImageNet-1K top-1 accuracy at 6.09 MB, while the ESP32-S3-EYE deployment uses the 224×224 FTerViT-Small model:

5.81 MB of packed weights in flash, 2.83 MB peak PSRAM, and 79.64% ImageNet-1K top-1 accuracy. A complete forward pass is reported at 21.06 s. The study also reports direct power measurements for selected kernel comparisons. This is important evidence for MCU-resident transformer execution, but it is a Vision Transformer rather than a language model and therefore should be treated as hardware/optimization evidence rather than proof that a general-purpose LLM can run on ESP32.

Table 3 compares the three included model-level optimization studies using resource and task context rather than ranking them by accuracy alone.

Table 3. Comparison of fully on-device model optimization studies.

STUDY	OPTIMIZATION	MODEL / TASK	ON-DEVICE FOOTPRINT	PERFORMANCE / LATENCY
Elumalai & Rajan (2026)	MPS decomposition + LSBA	Foundation models / predictive maintenance	Peak runtime SRAM	44–63× compression; 93.8–96.4% task accuracy; 19–142 ms on ESP32-S3; 1.4–4.7 mJ reported
Pujari et al. (2024)	Sparse ternary quantization + federated fine-tuning	Small language models / task-specific NLP	256 KB RAM	5.8× size reduction; ~92% of full-precision baseline accuracy; 4.3× latency reduction
Ruciński et al. (2026)	Full ternarization + distillation + QAT recovery	Vision Transformer / ImageNet-1K classification	ESP32-S3-EYE: 5.81 MB flash; 2.83 MB peak PSRAM	79.64% top-1 for 224×224 model; 21.06 s full forward pass; ~15× compression vs FP32 reference

The three optimization studies occupy different points in the design space and should not be ranked using their reported accuracy values because their tasks and baselines are different. MPS-TinyFM reports the largest compression ratio for specialized foundation-model workloads while remaining within approximately 389–412 KB of peak runtime SRAM. Pujari et al. target a smaller 256 KB RAM budget for SLM inference and report relative accuracy and latency improvements against their own full-precision baseline. FTerViT demonstrates a much larger fully on-device transformer using external PSRAM, but its 21.06 s end-to-end latency illustrates that model fit alone does not imply interactive performance. The value of FTerViT for this review is therefore architectural and computational: it shows that low-bit transformer kernels can execute fully on ESP32-S3 hardware when model storage and activation memory are carefully managed.

Taken together, the evidence supports a precise interpretation of “LLM on ESP32.” The reviewed literature demonstrates (i) practical LLM-enabled systems in which ESP32 acts as an endpoint or controller, (ii) locally hosted LLM control frameworks coupled to ESP32, and (iii) MCU-resident SLM, foundation-model, and Vision Transformer workloads under aggressive compression and task specialization. It does not demonstrate a general-purpose conversational LLM with open-ended text generation running entirely on an ESP32. This distinction should be preserved in future papers, which should explicitly report model class, parameter count, inference location, internal versus external memory use, and end-to-end latency.

5. DISCUSSION

The synthesis reveals a field with three distinct deployment classes rather than a single continuum of

“LLMs running on ESP32.” Cloud-assisted architectures place nearly all semantic computation on external services; local-host architectures retain the LLM on a nearby computer while using the ESP32 for deterministic execution; and fully MCU-resident studies compress transformer or small-language-model workloads until they fit within the microcontroller memory hierarchy. This taxonomy resolves a central ambiguity in the literature: an application can legitimately be described as LLM-enabled while still providing no evidence of LLM inference on the microcontroller. The strongest current evidence for true on-device execution concerns compact, specialized models rather than general-purpose conversational LLMs.

It is worth stating explicitly why an evidence base of only ten studies can still support the gaps identified here. The central claims of this review are existential rather than quantitative: that no included study demonstrates a general-purpose conversational LLM executing entirely on an ESP32, and that no shared benchmark or reporting standard exists for microcontroller-resident language inference. Such claims are established by the absence of qualifying evidence across a systematic search, not by the number of studies retrieved, and the deployment taxonomy - not the study count - carries the analytical weight.

Assisted architectures offer practical advantages because they allow developers to use much larger models than an MCU could host. Katuwal and Ghimire (2026), for example, report a 1.2 s application response in a cloud STT-LLM-TTS pipeline, while EdgeTalk-MCU reports ~0.15 s for the local host-side decision stage. These numbers should not be compared directly: they measure different pipeline boundaries, tasks, models, and physical systems. The appropriate conclusion is therefore architectural rather than

competitive. Cloud services maximize model access but introduce network and external-service dependencies; local-host systems can reduce cloud exposure and provide deterministic microcontroller control, but require a companion device. In both cases, the ESP32 contributes low-level sensing, communication, or actuation rather than executing the full LLM.

The fully on-device evidence demonstrates technical feasibility under substantially narrower conditions. MPS-TinyFM reports 44–63× compression with peak runtime SRAM of 389–412 KB for specialized foundation-model workloads; Pujari et al. report SLM inference within 256 KB RAM; and FTerViT demonstrates a 5.81 MB ternary Vision Transformer using 2.83 MB peak PSRAM on an ESP32-S3-EYE. These results show that transformer computation can be mapped to microcontroller-class hardware, but they also reveal different bottlenecks. Internal SRAM constrains compact language models, external-memory bandwidth becomes important for larger transformers, and end-to-end latency can remain high even when a model fits in memory. The evidence therefore supports task-specific on-device transformer inference, not unrestricted LLM capability.

From a model-design perspective, the included compression studies suggest that trained transformer representations can contain exploitable redundancy or low-rank structure. MPS decomposition exposes such structure in selected weight tensors, while ternary approaches reduce storage and arithmetic complexity by constraining weights to a very small discrete set. These results should not be generalized into the stronger claim that the information content of all foundation models is far below their parameter count. A more defensible interpretation is that certain model components and task-specific models tolerate substantial structured compression when capacity is allocated carefully. FTerViT further shows that components often left in full precision, such as patch embeddings and normalization parameters, can in some cases be quantized successfully when the training and operator design are adapted to the target hardware.

For practitioners, deployment choice should follow application constraints rather than a universal ranking. When an internet connection and external services are acceptable, cloud assistance offers access to larger models at the cost of network, privacy, and service dependencies. When data should remain local but a companion computer is available, local-host architectures such as EdgeTalk-MCU or Home Assistant-based systems provide a useful middle ground. Fully on-device inference is most attractive when offline operation, bounded communication dependency, or data locality is essential and the task can be handled by a compact model. Even then, design decisions should consider the complete memory hierarchy, including internal SRAM, flash, and

PSRAM, and should report end-to-end latency and measured power where possible. Safety claims should also remain scoped to the evaluated scenarios; for example, the 0% collision result in EdgeTalk-MCU supports the runtime shield under the reported tests but does not constitute a general safety guarantee.

This review has several limitations. The search was limited to English-language material available through five primary academic sources plus supplementary Google Scholar and was completed in June 2026. The inclusion criteria intentionally span both language-model systems and non-language transformer evidence when the latter provides direct ESP32-class deployment insight; this improves engineering coverage but increases heterogeneity. The evidence base is small (ten studies), includes multiple publication types, and is vulnerable to publication bias because unsuccessful deployment attempts are less likely to appear in the literature. For these reasons, the review does not perform a meta-analysis and avoids treating reported task accuracies as a common outcome variable.

The included studies also vary substantially in reporting completeness. Accuracy is expressed as command recognition, task-specific classification, relative retention against a baseline, ImageNet top-1 accuracy, or safety/collision outcomes. Latency may refer to model processing, host-side decision time, or full application response. Memory is variously reported as internal RAM, peak SRAM, model size in flash, or PSRAM use. Power reporting is particularly sparse; FTerViT provides direct power measurements for selected on-device kernel comparisons, whereas many other studies report no measured power data or provide only aggregate/derived energy values. This heterogeneity prevents a defensible statement that any one architecture is globally “best.”

A further limitation is task scope. The strongest fully MCU-resident evidence concerns predictive maintenance, task-specific SLM workloads, and image classification. These are fundamentally different from open-ended conversation, long-context reasoning, or general text generation. Assisted systems can access GPT-, Gemini-, DeepSeek-, or other server-class models, but their language capability originates outside the ESP32. No included study demonstrates a general-purpose conversational LLM running entirely on an ESP32-class microcontroller. Whether such capability becomes practical will depend on future progress in model architecture, quantization, memory systems, and hardware acceleration.

The heterogeneity itself suggests how future work should be organized. Studies should report a common set of deployment descriptors—model type and parameter count, inference location, internal and external memory, model storage, sequence or input size, latency boundary, energy/power measurement method, task metric, and software/runtime stack. With these

descriptors, researchers could begin to construct a Pareto frontier across accuracy or task utility, latency, memory, and energy rather than comparing isolated headline numbers from incompatible experiments.

First, future research should investigate scalable combinations of compression techniques for language models that remain meaningful after deployment on microcontrollers. Tensor decomposition, low-bit quantization, pruning, distillation, and architecture redesign are complementary rather than mutually exclusive. However, combinations should be evaluated end-to-end on the target MCU because a higher nominal compression ratio does not automatically translate into lower latency or energy if decompression, irregular sparsity, or external-memory traffic becomes the new bottleneck.

Second, the field needs a standardized benchmark for MCU-resident language inference. A useful benchmark should specify common hardware configurations (for example, ESP32-S3 variants with and without external PSRAM), a small set of representative language tasks, fixed input lengths, and standardized reporting of accuracy or task utility, latency, internal SRAM, PSRAM, flash footprint, and directly measured power. Existing TinyML benchmarks provide a useful methodological precedent, but a language-focused benchmark must explicitly capture token or sequence processing and the distinction between model-only and end-to-end system latency.

Third, hardware-software co-design should exploit features already available on ESP32-S3-class devices. The ESP32-S3 provides vector instructions for neural-network and signal-processing workloads, while studies such as FTerViT show the value of operators designed around low-bit arithmetic and memory constraints. Future work should investigate attention variants with linear or bounded memory growth, fused kernels, integer-only normalization, optimized KV/cache strategies for short-context language models, and runtime systems that minimize movement between flash, internal SRAM, and PSRAM.

Fourth, future studies should broaden the language tasks evaluated directly on microcontrollers. Current on-device evidence is dominated by classification or specialized inference. Useful next steps include intent detection, short-form command parsing, constrained question answering, anomaly explanation, and tightly bounded multi-turn interaction where context size is explicitly controlled. On-device personalization or federated adaptation may also be valuable, but its memory, privacy, and update costs must be reported separately from inference cost.

Fifth, more realistic system evaluation is needed. Assisted architectures should be tested under degraded network conditions and should report service and communication latency separately from local actuation time. MCU-resident models should be evaluated over

long-running workloads with measured power, thermal behavior, memory fragmentation, and variations in sensor/input quality. Field trials are particularly important because laboratory feasibility does not guarantee robust performance in always-on IoT deployments.

Finally, adaptive deployment is a promising bridge among the three classes identified in this review. A system could execute simple, privacy-sensitive, or time-critical tasks on the microcontroller, escalate more complex requests to a local host, and use the cloud only when local resources are insufficient and policy permits. Such systems should make escalation rules explicit and should measure the resulting trade-offs in latency, privacy exposure, energy, and failure modes. This three-tier perspective extends current host-microcontroller frameworks into a practical research direction for capable but resource-aware embedded language interfaces.

6. CONCLUSION

This systematic review examined ten studies identified through a PRISMA-guided search completed in June 2026 and found that the phrase “LLM on ESP32” covers three technically different situations: cloud-assisted LLM systems, local-host/edge-assisted LLM systems, and fully microcontroller-resident transformer or small-language-model inference. Cloud and local-host architectures can provide rich language capability because the large model executes elsewhere, while the ESP32 handles sensing, communication, state feedback, or deterministic actuation. Fully on-device studies demonstrate that specialized language/foundation-model and Vision Transformer workloads can execute on ESP32-class hardware under aggressive compression, with reported language/foundation-model RAM footprints of approximately 256–412 KB and model-size reductions up to 63×. However, these studies use heterogeneous tasks and benchmarks, and no included work demonstrates a general-purpose conversational LLM running entirely on an ESP32.

For system designers, the main implication is that deployment decisions should be based on inference location and application constraints rather than headline accuracy alone. Cloud assistance is appropriate when connectivity and external-service dependence are acceptable; local-host architectures provide a middle ground for private or low-latency environments with a companion computer; and fully on-device inference is most suitable for narrow tasks where offline operation, data locality, or bounded communication dependency justifies severe model compression. Reported results should always distinguish model latency from end-to-end system latency and internal SRAM from external-memory use.

Future progress will depend on standardized MCU-language benchmarks, stronger reporting of memory

and power, scalable combinations of compression methods, and hardware-software co-design that exploits low-bit arithmetic and ESP32-S3 vector capabilities. Equally important, future publications should state explicitly whether the language model runs in the cloud, on a local host, or on the microcontroller. With this distinction in place, the field can move from loosely defined “LLM-enabled ESP32” demonstrations toward reproducible engineering evidence for genuinely capable, private, and real-time intelligence on resource-constrained devices.

REFERENCES

- Bertoletti, P. (2019). Proyectos con ESP32 y LoRa. Marcombo.
- Brown, T. B., Mann, B., Ryder, N., et al. (2020). Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33, 1877–1901.
- Chirath, M. H. D. (2025). A handwriting-based IoT control system for speech-impaired users using LLMs and edge devices. *Journal of Research Technology and Engineering*, 6(3), 38–47.
- Elumalai, D. K. and Rajan, R. (2026). MPS tensor compression for foundation model deployment on TinyIoT edge devices. 2026 IEEE World AI IoT Congress (AIoT), 239–244.
- Espressif Systems. (2026a). ESP-IDF Programming Guide: Performance optimization for ESP32 and ESP32-S3. Espressif Systems.
- Espressif Systems. (2026b). ESP32-S3 Series Datasheet, Version 2.2. Espressif Systems.
- Gupta, E. and Gupta, V. (2025). A comparative analysis of ESP32 and ESP8266 for AI-powered applications. 2025 International Conference on Next Generation Communication & Information Processing (INCIP), 992–996. <https://doi.org/10.1109/INCIP64058.2025.11019649>.
- Han, S., Pool, J., Tran, J. and Dally, W. (2015). Learning both weights and connections for efficient neural networks. *Advances in Neural Information Processing Systems*, 28.
- Hinton, G., Vinyals, O. and Dean, J. (2015). Distilling the knowledge in a neural network. *arXiv:1503.02531*.
- Jacob, B., Kligys, S., Chen, B., Zhu, M., et al. (2018). Quantization and training of neural networks for efficient integer-arithmetic-only inference. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2704–2713.
- Katuwal, B. and Ghimire, A. (2026). Voice-controlled smart home automation: Integrating large language models with IoT devices for enhanced usability. *International Research Journal of Engineering and Technology*, 13(1).
- Ouyang, L., Zhu, G. and Zhang, Y. (2025). Design of DeepSeek big-model intelligent voice assistant based on ESP32 microcontroller. 2025 IEEE International Conference on Pattern Recognition, Machine Vision and Artificial Intelligence (PRMVAI), 1–5. <https://doi.org/10.1109/PRMVAI65741.2025.11108406>.
- Page, M. J., McKenzie, J. E., Bossuyt, P. M., et al. (2021). The PRISMA 2020 statement: An updated guideline for reporting systematic reviews. *BMJ*, 372, n71.
- Pikus, M., Waś, J. and Kozina, A. (2026). Integration of Polish large language models for smart home automation. In *Advances in Artificial Intelligence Research: PP-RAI 2025, Lecture Notes in Networks and Systems*, 1599, 241–251. Springer. https://doi.org/10.1007/978-3-032-04197-5_19.
- Pujari, M., Goel, A. and Pakina, A. K. (2024). Efficient TinyML architectures for on-device small language models: Privacy-preserving inference at the edge. *International Journal Science and Technology*, 3(3), 67–75. <https://doi.org/10.56127/ijst.v3i3.1958>.
- Ruciński, S., Bonazzi, P., Türetken, E., Narduzzi, S., Magno, M. and Maamari, N. (2026). FTerViT: Fully ternary vision transformer. *arXiv:2605.21171*.
- Ščulac, R. (2023). GPT virtualni assistant implementiran na ESP32 platformi [Thesis]. University of Rijeka, Faculty of Engineering.
- Vaswani, A., Shazeer, N., Parmar, N., et al. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, 30.
- Warden, P. and Situnayake, D. (2019). *TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers*. O’Reilly Media.
- Xiong, J. and Bao, J. (2026). EdgeTalk-MCU: State-aware prompt-constrained local LLM control with runtime shielding for low-latency microcontroller interaction. *Applied Sciences*, 16(12), 5748. <https://doi.org/10.3390/app16125748>.
- Zhang, Y., Suda, N., Lai, L. and Chandra, V. (2017). Hello edge: Keyword spotting on microcontrollers. *arXiv:1711.07128*.
- Ziegeldorf, J. H., Morchon, O. G. and Wehrle, K. (2014). Privacy in the Internet of Things: Threats and challenges. *Security and Communication Networks*, 7(12), 2728–2742.