Going Beyond the Edge: Distributed Inference of Transformer Models on Ultra-Low-Power Wireless Devices

Alexander Gräfe[1] , **Ding Huo**[2] , **Vincent de Bakker**[1] , **Johannes Berger**[1] , **Marco Zimmerling**[2] , **Sebastian Trimpe**[1]
[1]RWTH Aachen University
[2]TU Darmstadt
alexander.graefe@dsme.rwth-aachen.de

Abstract

Transformer models are rapidly becoming a cornerstone of modern Internet of Things (IoT) applications, yet their computational and memory demands far exceed the capabilities of a single typical ultra-low-power IoT device. We present CATS, a framework for distributed transformer inference on ultra-low-power wireless devices, enabling multiple devices to collaboratively execute models far larger than what a single device can sustain. At its core, CATS is a communication-aware distributed transformer inference scheme co-designed across transformer partitioning, wireless communication and training. It employs SomeGather, a new pruned communication primitive that selectively broadcasts activation columns to reduce communication bandwidth and RAM usage without sacrificing model accuracy. Building on SomeGather, we design a partitioning method that exploits this primitive for efficient model parallelism. To cope with unreliable wireless communication, CATS employs message-dropout during training, which mimics packet losses and yields models that are robust to message loss during inference. In real-world experiments, we show that CATS brings distributed transformer inference to ultra-low-power wireless devices for the first time, with deployments on up to 16 devices that collaboratively execute transformer models up to 14 times larger than what a single device can run.

1 Introduction

Transformer models have become the backbone of intelligent IoT systems, delivering state-of-the-art performance in natural language processing, computer vision and time-series analysis [Kaur and Jadhav, 2023; Zong *et al.*, 2025; Zhang *et al.*, 2025b]. This remarkable capability, however, comes at a substantial computational cost and typically requires powerful hardware, which conflicts with the tight resource budgets of IoT downstream devices like smart sensors.

To address these challenges and enable scalable deployment of large transformer models on downstream devices, recent work has identified distributed transformer inference (DTI) as a promising paradigm [Liu *et al.*, 2025b; Bochem *et al.*, 2025; Hu and Li, 2024; Zhang *et al.*, 2025a; Liu *et al.*, 2025a]. In many applications, such as sensor networks, devices naturally operate in groups. By pooling resources from multiple devices, DTI can execute transformers much larger than what a single device can handle, thereby unlocking advanced transformer models in highly resource-constrained environments.

Existing approaches, however, target powerful platforms (e.g., Raspberry Pis or specialized microcontroller units (MCUs) with tensor processing units) connected via high-speed wired links. In contrast, the vast majority of real-world deployments, such as wireless sensor networks for environmental monitoring, infrastructure or agriculture, are far more constrained [Sanislav *et al.*, 2021; Rahaman and Azharuddin, 2022; Sofi *et al.*, 2022]. First, they typically rely on small batteries or energy harvesting as a power source and can thus only support ultra-low-power devices, which consist of MCUs with very limited memory and compute power [Borges *et al.*, 2014; Jamshed *et al.*, 2022], several orders of magnitude lower than that of e.g., a Raspberry Pi. Second, devices in such applications communicate via low-power wireless interfaces such as Bluetooth Low Energy (BLE), forming mesh networks to cover large areas, enable flexible deployment and support mobile nodes [Zimmerling *et al.*, 2020; Baumann *et al.*, 2020; Chai *et al.*, 2024]. While creating unprecedented opportunities for smart IoT systems, this setting comes with several fundamental challenges for DTI, which are still unexplored:

- **C1 Limited Computing Power** DTI of transformers on resource-constrained MCUs is bottlenecked by all key hardware resources: scarce flash memory for storing weights, limited RAM for intermediate activations and constrained compute throughput for timely inference.
- **C2 Limited Communication Bandwidth** Low-power wireless DTI is constrained by low-bandwidth, high-latency inter-device links that turn communication into a dominant bottleneck. Mesh networks further exacerbate this challenge as packets are forwarded over multiple hops.
- **C3 Unreliable Communication** As wireless communication is inherently unreliable, communicated intermediate activations may be lost, thereby significantly degrading accuracy.

These constraints render existing DTI approaches [Liu *et al.*, 2025b; Bochem *et al.*, 2025; Hu and Li, 2024; Zhang *et al.*, 2025a; Liu *et al.*, 2025a] largely unsuitable in

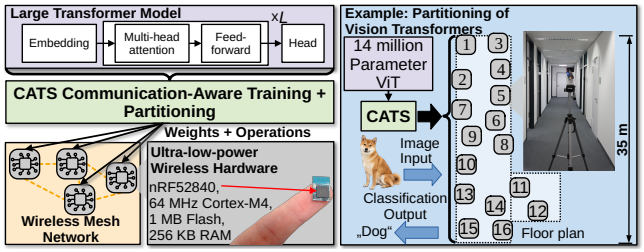

Figure 1: Collaborative Inference at the Sensor-level (CATS). CATS contains a communication-aware training and partitioning scheme that enables the execution of large transformer networks to be parallelized across ultra-low-power wireless devices communicating in mesh networks. Our resulting hardware implementation consists of 16 nRF52840 devices (64 MHz Cortex-M4, 1 MB flash, 256 kB RAM) spread on a floor of a university building, communicating over a two-hop network on BLE's physical layer. This implementation is able to execute a 14 million parameter Vision Transformer (ViT), fourteen times larger than what a single device could execute.

the ultra-low-power wireless setting. Many methods overlook critical resource bottlenecks (**C1**). Some reduce RAM usage but leave flash limits unaddressed [Liu *et al.*, 2025b; Hu and Li, 2024], while others do the opposite [Bochem *et al.*, 2025]. Communication constraints (**C2**) are only partially tackled, often by assuming specific network structures, such as grouped device configurations [Bochem *et al.*, 2025], which are unavailable in arbitrary mesh networks and dynamic network topologies. Moreover, all approaches assume lossless communication, which may hold in wired settings but fails in wireless networks (**C3**) [Srinivasan *et al.*, 2010]. As a result, no method for DTI on ultra-low-power wireless devices exists today, exposing a substantial gap between current DTI research and the demands of real-world IoT deployments.

Contributions. To close this gap, we introduce CATS (***C**ollaborative Inference **at** the **S**ensor-level*) (Figure 1). Our key idea is to design the transformer partitioning, communication and training jointly around a pruned all-to-all primitive called SomeGather. By pruning entire activation columns in a communication-aware way, SomeGather turns a dense transformer into one where feature subspaces are processed purely locally, while a carefully chosen subset is shared. This simultaneously cuts communication volume and reduces the activations each device must store in RAM (addresses **C1** and **C2**) while maintaining accuracy. As a result, our partitioning scheme based on SomeGather allows reducing the per-device RAM, flash and computing load at the same time (**C1**), unlike existing approaches. To cope with stochastic message losses (**C3**), we inject message-dropout (MD) during training, a structured dropout exposing the model to realistic loss patterns making it robust to message loss during inference.

As a result, CATS enables efficient collaborative execution of large transformer models across multiple ultra-low-power wireless devices. Figure 1 shows CATS in action on 16 MCUs communicating via a low-power wireless mesh. They jointly execute a 14 million parameter Vision Transformer (ViT), over $14\times$ larger than fits on a single device. To the best of our knowledge, this is the first demonstration of DTI on ultra-low-power wireless hardware, showing that CATS substantially cuts per-device resource requirements and makes large transformers practical on such platforms. In contrast, current state-of-the-art approaches [Hu and Li, 2024; Liu *et al.*, 2025b; Bochem *et al.*, 2025] deployed on the devices would only execute transformers no larger than what a single device can run. Experiments further reveal that SomeGather reduces communication volume by up to $90\,\%$ and per-device activation RAM by $67.5\,\%$ compared to unpruned DTI, while maintaining accuracy on four time-series prediction benchmarks and MD reduces the relative prediction error increase from none to $10\,\%$ message loss from up to $200\,\%$ to $23.6\,\%$ across datasets.

To summarize, our contributions are:

1. A new communication primitive, SomeGather, and a partitioning scheme for transformer models that express all cross-device communication as column-pruned all-to-all exchanges, simultaneously reducing communication, per-device computing load, RAM and flash footprint.
2. MD, a structured dropout mechanism that simulates realistic message-loss patterns across all transformer layers, improving robustness to lossy communication.
3. The first end-to-end demonstration of DTI on ultra-low-power wireless devices.

2 Background – Transformer Networks

This section introduces the notation for transformer models used in this work. For brevity, we focus on encoder- and decoder-only architectures, which are state-of-the-art in domains such as vision and time-series processing [Dosovitskiy *et al.*, 2021; Nie *et al.*, 2023]. A transformer maps tokens X_1 (e.g., image or time-series patches [Dosovitskiy *et al.*, 2021; Nie *et al.*, 2023]) to new tokens X_T via T transformer layers. Each layer i consists of an attention and a residual block:

$$\bar{X}_i = \text{layernorm}(X_i) \tag{1}$$

$$Q_i = \bar{X}_i W_{i,Q}, \quad K_i = \bar{X}_i W_{i,K}, \quad V_i = \bar{X}_i W_{i,V} \tag{2}$$

$$H_h = \text{softmax}\left(\frac{Q_{i,h} K_{i,h}^T}{\sqrt{F/H}}\right) V_{i,h}, \quad h \in \{1, \dots, H\} \tag{3}$$

$$Y_i = [H_1, H_2, \dots, H_H] W_{i,O} + X_i \tag{4}$$

$$X_{i+1} = W_{i,L} f(\dots W_{i,2} f(W_{i,1} \text{layernorm}(Y_i))) + Y_i. \tag{5}$$

The transformer block first applies row-wise layernorm to its input X_i to obtain $\bar{X}_i$ and then linearly projects it into queries, keys and values $Q_i, K_i, V_i \in \mathbb{R}^{N \times F}$ using weight matrices $W_{i,Q}, W_{i,K}, W_{i,V}$ (biases omitted). They are split into H parts $Q_{i,h}, K_{i,h}, V_{i,h} \in \mathbb{R}^{N \times (F/H)}$, from which attention heads produce H_h. These are concatenated, projected with weights $W_{i,O}$ and combined with input X_i via a residual connection to obtain the output Y_i. A residual block with L layers, input layernorm, element-wise activation function f and weights $W_{i,\ell}$ then leads the output X_{i+1}.

3 Related Work

To our knowledge, no existing methods support DTI on ultra-low-power wireless devices. Current approaches target substantially more powerful edge platforms such as Raspberry

Pis [Wei *et al.*, 2024; Liu *et al.*, 2025a; Wen *et al.*, 2025], Nvidia Jetsons [Xu *et al.*, 2023], laptops [Liu *et al.*, 2025b] and virtual machines [Hu and Li, 2024], and assume wired communication. The only study employing MCUs [Bochem *et al.*, 2025] uses Siracusa chips [Prasad *et al.*, 2024] with multiple RISC-V cores, again connected via wired links. In contrast, CATS targets cost-effective, common wireless MCUs, enabling DTI in an unaddressed regime. Given the lack of complete end-to-end solutions, we therefore review prior work that addresses individual aspects relevant to CATS.

3.1 Partitioning Strategies for Transformers

DTI methods for edge devices differ in their partitioning strategies and which communication primitives they use for them.

Communication primitives. Most DTI methods build on three communication primitives to distribute matrix multiplications: AllGather, AllReduce and ReduceScatter [Stahl *et al.*, 2021; Du *et al.*, 2024]. Using AllGather, devices initially hold disjoint parts of a matrix and broadcast them so that, afterwards, each device holds the entire matrix. Using AllReduce/ReduceScatter, devices hold partial summands of a global sum and, through collaborative communication, end up with the result of the sum, either the entire matrix (AllReduce) or only parts (ReduceScatter). Efficient implementations of AllReduce/ReduceScatter, however, typically assume specialized, structured communication topologies (e.g., groups [Bochem *et al.*, 2025]), and become RAM- and communication-inefficient in unstructured mesh networks, as we later quantify in Section 5.2.

Partitioning strategies. Recent DTI strategies for edge devices build on these primitives and can be classified into three main categories. The first class [Liu *et al.*, 2025b; Wen *et al.*, 2025; Hu and Li, 2024] splits the transformer along the token dimension and communicates during self-attention. This approach improves computational throughput and reduces RAM usage but still requires each device to store the full set of weights, leaving the flash footprint unchanged.

The second class partitions the transformer along the feature dimension and distributes different attention heads across devices [Bochem *et al.*, 2025]. This lowers flash memory requirements because each device only stores a subset of the weights. However, this approach relies on the communication- and RAM-heavy AllReduce primitive.

The third class assigns smaller, independent models to each device and then merges their outputs [Xu *et al.*, 2023; Liu *et al.*, 2025a], or decouples transformer components [Wei *et al.*, 2024]. These methods are either restricted to image-classification transformers or ultimately require merging blocks and intermediate results using the previously mentioned strategies, thereby inheriting similar limitations.

Positioning CATS. CATS builds on the strengths of the second class by dividing the transformer along the feature dimension and distributing attention heads across devices, reducing per-device flash memory usage. To avoid the communication and RAM overhead of AllReduce in mesh networks, CATS relies on SomeGather. SomeGather is a pruned variant of AllGather that broadcasts only a selected subset of activation columns, reducing both communication volume and per-device RAM compared to pure AllGather. As Sections 4.2 and 4.3 detail, this tailored design simultaneously lowers flash usage, RAM requirements and per-device computational load, making it well suited for ultra-low-power wireless platforms.

3.2 Communication-Aware Distributed Inference

Current DTI approaches do not consider constrained and unreliable mesh communication. However, methods for other types of Neural Networks (NNs) address parts of these challenges.

Communication-aware pruning. Specialized pruning techniques [Mao *et al.*, 2017; Abdi *et al.*, 2020; Jian *et al.*, 2023; Qin *et al.*, 2023] reduce communication latency by removing transmitted NN connections. While prior work predominantly targets convolutional NNs, CATS extends this idea to transformers. Moreover, we demonstrate that such pruning not only lowers communication overhead but can also reduce RAM requirements, an aspect that is overlooked in existing communication-aware pruning strategies.

Distributed inference over lossy networks. Current work on inference under message loss focuses on a two-device, server-edge setting [Itahara *et al.*, 2022; Cheng *et al.*, 2024; Hou and Ohtsuki, 2025], where the edge executes the front part of an NN and the server the back part. These approaches show that simulating lossy communication during training via tailored dropout between the front part and the back part significantly improves robustness at inference time. CATS generalizes this concept to distributed inference across multiple devices, where the execution of each layer is shared among them. We introduce MD that applies dropout simulating lossy communication to *all* layers, going beyond a single split point.

Distributed inference in mesh networks. Existing approaches explicitly consider the network graph either in a task-allocation problem [Samikwa *et al.*, 2023; Jung and Lee, 2023; Disabato *et al.*, 2021] or during training [Jian *et al.*, 2023]. In realistic wireless settings, the network topology is dynamic, influenced by environmental changes or device mobility (e.g., wearable devices) [Hao *et al.*, 2025]. As a result, these approaches require repeated optimization and allocation steps during changes, which incurs substantial overhead. CATS follows a different strategy by implementing a communication scheme that abstracts away the details of mesh connectivity while providing properties that are particularly advantageous for DTI (see Section 4.5).

4 CATS – Distributed Transformer Inference on Ultra-Low-Power Wireless Devices

This section presents how CATS enables DTI on ultra-low-power wireless devices by minimizing per-device RAM, flash and compute (**C1**) while addressing limited and unreliable mesh communication (**C2**, **C3**). We first outline all components and their interactions, then detail each component.

4.1 Overview

We consider D devices in a wireless mesh network with a time-varying topology to accommodate mobile devices like wearables. The devices must execute a transformer for data analysis. We focus on distributing the transformer layer computation

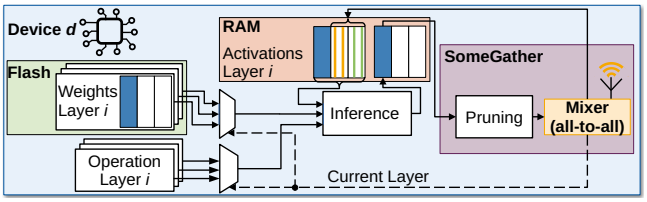

Figure 2: Device Operation. Devices operate in synchronized rounds of communication and computation. The SomeGather communication primitive prunes the transmitted data, to reduce bandwidth and RAM demands. Each device stores only a part of the activations and weights, reducing RAM and flash usage.

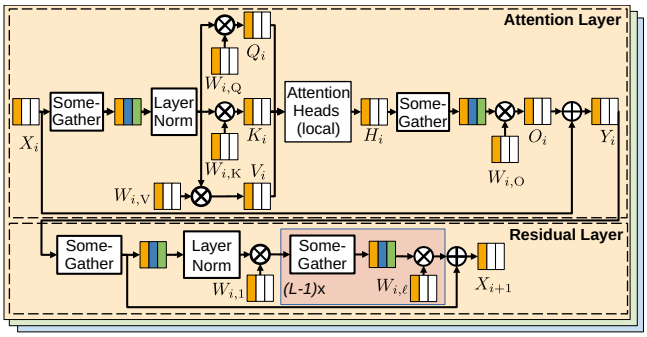

Figure 3: Partitioning Scheme of CATS. Our partitioning scheme is based on SomeGather and input partitioning of matrix multiplication. The multi-head attention mechanism is split along the attention heads, with each device calculating different attention heads.

(Equations (1)–(5)). Data sources and sinks are application-specific and thus out of scope, but our approach generally supports single as well as distributed sources and sinks.

At a high level, CATS organizes inference into tightly synchronized rounds of communication and computation. In each round, every device first computes its share of the current layer's operations on its locally stored activations and weights as shown in Figure 2. The devices then exchange intermediate results using SomeGather, a pruned version of AllGather that broadcasts only a selected subset of activation columns instead of all of them. This allows us to reduce the volume of transmitted data and keep the per-device RAM for activations small. On top of this, we design a partitioning scheme that distributes the transformer model across devices such that all inter-device exchanges can be expressed in terms of SomeGather operations. Finally, CATS uses MD, a mechanism that stochastically removes activations according to Mixer's loss characteristics, making the model robust to packet loss at inference time. To realize SomeGather efficiently over a wireless mesh, CATS relies on the Mixer protocol [Ferrari *et al.*, 2011; Herrmann *et al.*, 2018; Zimmerling *et al.*, 2020], which provides all-to-all communication with low latency and high reliability despite dynamic link conditions thus allowing us to abstract mesh communication away during partitioning.

4.2 SomeGather

We illustrate SomeGather on a distributed matrix multiplication $B = AW$ with column-split matrices. We consider square matrices A and W with an equal partitioning across devices, i.e., each device holds S columns of A, W and B. Extensions to non-square matrices or uneven partitionings follow directly by adapting the respective dimensions. The next section discusses application of SomeGather to transformers.

In a conventional AllGather-based implementation, devices first exchange all columns of A. Each device then multiplies its local copy of A with its local columns of W, yielding a subset of the columns of $B = AW$ on every device [Stahl *et al.*, 2021]. In this setting, RAM scales as $\mathcal{O}(C_\mathrm{A} + C_\mathrm{W}/D)$ and the communication scales as $\mathcal{O}(C_\mathrm{A})$, where C_A and C_W denote the sizes of A and W. This is problematic on ultra-low-power devices, because C_A can be large.

SomeGather addresses this bottleneck by pruning communication of activations. To support arbitrary sequence lengths N, pruning individual rows (i.e., timestamps) is ineffective. We thus prune entire activation columns and broadcast only some columns of A instead of all. Non-transmitted columns incur no inter-device traffic and are processed purely locally, reducing both the RAM required for received activations and the overall communication volume.

Mixer's all-to-all behavior constrains how we can prune. Whenever one device transmits, all others receive the same packet, which precludes pruning individual links between specific pairs of devices, as in prior schemes [Mao *et al.*, 2017; Abdi *et al.*, 2020; Jian *et al.*, 2023; Qin *et al.*, 2023]. Pruning must therefore remove all outgoing connections from one device to all other devices. Formally, each device d maintains a binary matrix $p_d \in \{0,1\}^{S\times 1}$, where an entry of 1 indicates that the corresponding activation column of A is transmitted. The pruned multiplication can then be expressed as $B = A(W \odot P)$, with Hadamard product $\odot$ and

$$P = \begin{bmatrix} 1_{S\times S} & (p_1)_{\times S} & \cdots & (p_1)_{\times S} \\ (p_2)_{\times S} & 1_{S\times S} & \cdots & (p_2)_{\times S} \\ \cdots & \cdots & \cdots & \cdots \\ (p_D)_{\times S} & (p_D)_{\times S} & \cdots & 1_{S\times S} \end{bmatrix}. \quad (6)$$

where $1_{n\times m}$ is an $n \times m$ matrix of ones and $(p_d)_{\times S}$ denotes repeating p_d S times column-wise. The diagonal blocks of P are ones, preserving connections within a device that require no communication. The off-diagonal blocks are defined by p_d, which removes all connections from one device to another. The mask P is not used during inference but during training.

Training with pruning is performed stepwise as in [Qin *et al.*, 2023]. At each stage, we prune a fixed fraction of neurons in every layer and then retrain. Columns are ranked by the sum of absolute values of all outgoing weights and those with the smallest sums are pruned.

4.3 Partitioning Strategy on Top of SomeGather

Based on this SomeGather primitive, we design a partitioning strategy that relies on it exclusively. CATS distributes each multi-head attention layer by assigning every device a disjoint subset of the H attention heads and splitting all activations and weights along the feature (column) dimension, as illustrated in Figure 3. Each layer starts from an input matrix X_i whose columns are partitioned across devices. In the first SomeGather operation of the layer, each device receives columns of X_i from all other devices. The subsequent

layernorm requires some adjustments: standard layernorm computes the mean and standard deviation over each complete row, but with pruning no single device necessarily holds all columns of a row. We therefore use device-local layernorm, where each device computes the mean and standard deviation from only its activation columns.

After device-local layernorm, devices calculate queries, keys and values locally, using only their assigned columns of $W_{i,Q}, W_{i,K}, W_{i,V}$. We split the weights such that each device calculates exactly those columns of Q_i, K_i, V_i required by its assigned self-attention heads. Consequently, head-specific computations (Equation (3)) are executed fully locally, without further inter-device communication.[1] After concatenating their outputs along columns, each device holds distinct columns of H_i. Those are partially exchanged via a SomeGather operation. Afterwards, devices multiply it with their assigned columns of $W_{i,O}$ yielding column-wise partitioned O_i. We partition $W_{i,O}$ so that column indices align with those of the input X_i, enabling local addition of the residual connection.

The subsequent residual block (Equation (5)) follows the same partitioning scheme. In each layer ℓ, devices perform a SomeGather operation followed by multiplication with their designated columns of $W_{i,\ell}$, producing an output also partitioned along columns, serving as input for the next similarly partitioned layer. Thus, the entire block's output X_{i+1} is column-partitioned such that it is compatible with the subsequent attention layer's requirements.

4.4 Message Dropout

MD is used to simulate message loss during training. For Mixer, but also all-to-all protocols in general, we distinguish three modes of message loss: (a) a device does not receive the message from another device, (b) a device does not receive messages from any other device, (c) all other devices do not receive the message from one device. During training, CATS simulates these three modes by sampling three weight masks. These masks share a similar structure of the pruning matrix P, but are resampled at every training step. MD must account for transformer-specific behavior, i.e., the weight masks for computing queries, keys and values must be identical, and training performs separate layernorm operations for each device. By sampling such masks at every training step, CATS exposes all layers to realistic patterns of missing messages, which improves robustness under lossy wireless communication.

4.5 Mixer

CATS relies on efficient broadcast in wireless mesh networks. For our hardware implementation, we therefore employ Mixer [Herrmann *et al.*, 2018]. Mixer has proven to work under rapid device movement and furthermore offers order-optimal scaling, which is crucial for low communication latency. Consequently, [Gräfe *et al.*, 2026] already demonstrated Mixer's suitability for distributed learning.

Mixer combines synchronous transmissions with random linear network coding to efficiently flood messages through the network [Zimmerling *et al.*, 2020; Ho *et al.*, 2006]. In each SomeGather round, Mixer operates in discrete time slots with microsecond-level synchronization via phase-locked loops, enabling devices to exploit the capture effect [Leentvaar and Flint, 1976]. This effect allows receivers to decode the strongest among simultaneous transmissions, substantially reducing scheduling complexity and communication overhead compared to protocols that avoid concurrent transmissions. Using network coding, devices transmit linear combinations of their own messages and previously received data, rapidly disseminating information through the network. By solving the resulting system of equations at the end of each round, all devices can recover all transmitted messages.

5 Evaluation

This section presents an in-depth evaluation of CATS and compares it to the state-of-the-art.

5.1 Evaluation Goals and Methodology

We evaluate whether CATS addresses challenges **C1**–**C3** by structuring our study around four key questions:

- **Q1** How does CATS scale regarding memory (**C1**)?
- **Q2** How does CATS scale regarding latency (**C1** and **C2**)?
- **Q3** Does pruning inside SomeGather preserve model accuracy (**C1** and **C2**)?
- **Q4** Can MD make models robust against message loss (**C3**)?

To answer these questions, we combine hardware experiments with simulation studies, providing both realistic validation on actual devices and coverage of a much broader configuration space than feasible on hardware alone. [2]

Testbed. We perform hardware experiments on a testbed consisting of up to 16 devices (nRF52840, 64 MHz Cortex-M4, 1 MB Flash, 256 KB RAM) placing them along a $35\,\mathrm{m}$ long hallway of a university building leading to a network diameter of at least two hops (see Figure 1). The testbed utilizes the original Mixer implementation [Herrmann *et al.*, 2018] using the nRF52840's BLE physical layer. For layer computations, we use 8-bit kernels provided by CMSIS-NN [Lai *et al.*, 2018].

Model and Dataset. Our simulation study uses time-series transformers [Nie *et al.*, 2023] as an exemplary application. The models have six transformer layers with a feature dimension of 128 and one hidden layer in the residual blocks. We train the networks on the ETT-h2 [Zhou *et al.*, 2021], ICD [Von Birgelen *et al.*, 2018], London-smart-meters and Traffic [Godahewa *et al.*, 2021] datasets. Training uses ADAM [Kingma and Ba, 2015] with a batch size of 2048 and a learning rate of 0.001 with a cosine schedule over 50 epochs. All runs are repeated ten times with different random seeds.

Metrics. We focus on four key system metrics. RAM usage and flash footprint for **Q1** are obtained analytically. Latency for **Q2** is measured directly on hardware. Accuracy for **Q3** and **Q4**, reported as mean square prediction error, is derived from simulation, since exhaustive hardware evaluation across all models and datasets is infeasible.

[1] In our implementation, we calculate the heads and their inputs $Q_{i,h}, K_{i,h}, V_{i,h}$ sequentially to save RAM.

[2] Code available at: github.com/Data-Science-in-Mechanical-Engineering/CATS.

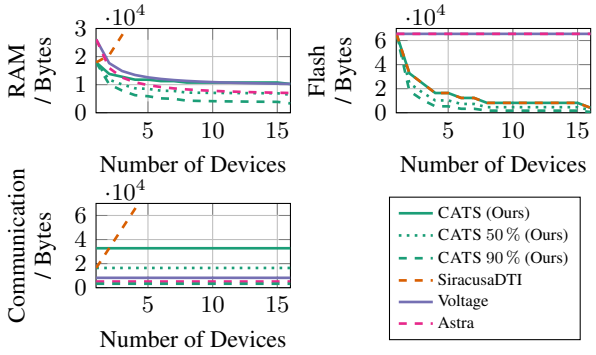

Figure 4: Resource Usage Comparison Across Different Approaches. Values following CATS indicate the pruning percentage applied within SomeGather. CATS consistently achieves simultaneous reductions in both RAM and flash usage, whereas all baseline methods fail to reduce both resources at once.

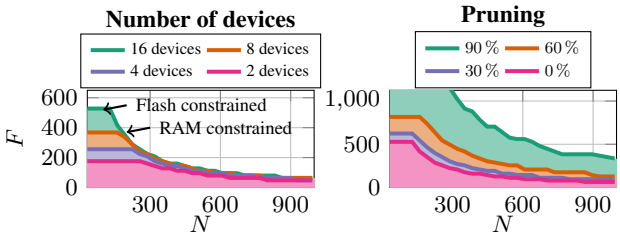

Figure 5: Model Size Scaling. Colored regions indicate feasible combinations of feature size F and token count N for each configuration. For small N, flash capacity bounds F, whereas for larger N the limit is set by RAM. Left plot: Scaling with the number of devices without pruning. Adding devices increases the attainable feature size by decreasing flash usage per device, while RAM requirements grow only marginally. Right plot: Scaling with pruning. Higher pruning ratios (legend) markedly extend the supported token count N, as SomeGather substantially reduces the RAM footprint.

Baselines. As CATS is, to our knowledge, the first method enabling DTI on ultra-low-power wireless devices, no end-to-end baseline exists. We therefore use question-specific baselines. For **Q1**, we benchmark our partitioning strategy against Voltage [Hu and Li, 2024], Astra [Liu *et al.*, 2025b] and SiracusaDTI [Bochem *et al.*, 2025], assuming their integration into our Mixer-based framework. For **Q2**, we compare CATS-based distributed execution with centralized execution on a single device. For **Q3**, we compare SomeGather with AllGather and normal pruning, pruning an AllGather-distributed transformer to match the bandwidth of a pruned SomeGather model. For **Q4**, we quantify robustness to message loss by comparing models trained with and without MD.

5.2 Q1 – Memory Scaling

We assess **Q1** in two experiments. First, we benchmark against established baselines. Second, we determine the maximum transformer size that our implementation can execute.

Baseline Comparison

Figure 4 compares CATS against the baselines Astra, Voltage and SiracusaDTI. As the number of devices grows, both Astra and Voltage reduce per-device RAM. Voltage consistently uses more RAM than CATS, while Astra undercuts unpruned CATS as it relies on specialized low-precision quantization. With pruning inside CATS, RAM drops below Astra. In contrast, SiracusaDTI's RAM usage grows linearly with the number of devices due to its AllReduce-based partitioning, which is poorly supported in mesh networks.

Flash utilization for both Voltage and Astra remains constant, as the weights are not distributed across devices. In contrast, SiracusaDTI and CATS distribute all weights evenly across devices, resulting in similar Flash usage that systematically decreases with an increasing number of devices.

Without pruning, CATS transmits more data than Voltage and Astra. However, by increasing the pruning ratio inside SomeGather, the communication volume can be reduced to comparable levels. In contrast, SiracusaDTI's communicated data grows linearly with the number of devices due to its reliance on the AllReduce-operation.

Overall, no existing baseline reduces both RAM and flash: Voltage and Astra reduce RAM, whereas SiracusaDTI targets flash. In contrast, CATS simultaneously reduces both and its higher communication cost is largely mitigated by pruning in SomeGather, making CATS highly resource-efficient.

Model Size Scaling

Figure 5 quantifies scaling to feature size F and token count N. For each configuration, we compile the model. If it fails, flash or RAM limits are exceeded. For small N, flash capacity bounds F, whereas for large N, RAM becomes the dominant bottleneck, since weight flash usage scales quadratically with F, while RAM scales only linearly with N. Increasing the number of devices substantially relaxes flash constraints by distributing the weights onto more devices, while RAM constraints remain almost unchanged because unpruned SomeGather (i.e., AllGather) scales only marginally with the number of devices. Conversely, increasing pruning in SomeGather reduces flash and RAM constraints, as devices store fewer activations in RAM and fewer weights in flash.

In summary, CATS significantly improves model size scalability, with SomeGather as the key mechanism enabling efficient utilization especially of RAM.

5.3 Q2 – Latency Scaling

Using the hardware implementation of CATS, we evaluate the latency of attention and residual blocks (Figure 6). We first focus on non-pruned inference (solid bars in Figure 6). Computing latency decreases with more devices, whereas communication latency remains nearly constant (with a slight increase for 16 devices and 256 features) and dominates runtime, ranging from $69.42\,\%$ (attention block with four devices) to $91.4\,\%$ (residual block with 16 devices). Despite this, CATS achieves a speedup of $1.27\times$ for the attention block at 256 features and $2.51\times$ at 512 features compared to central execution. For the residual block, CATS shows a slowdown at 256 features ($0.54\times$), but reaches a speedup of $1.38\times$ at 512 features.

Pruning via SomeGather markedly enhances scalability. With a pruning ratio of $90\,\%$, communication latency is reduced by up to $80\,\%$. For 16 devices, increasing pruning from

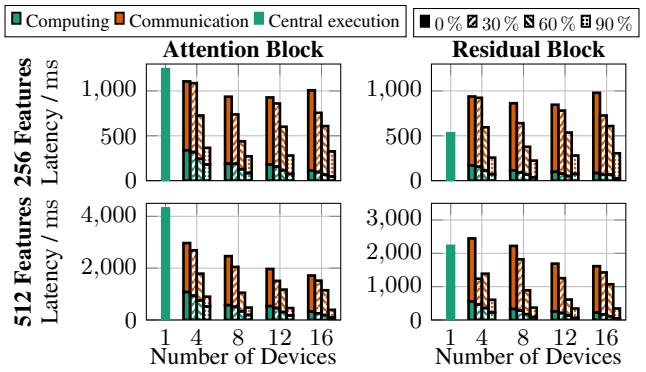

Figure 6: Latency of CATS Versus Number of Devices. We measure latency for attention and residual blocks for 256 and 512 features. Different bar patterns show different SomeGather pruning ratios. Central execution latency for 512 features is estimated by multiplying the computing latency at four devices by four as the layers do not fit on a single device. Computing latency decreases with number of devices while communication latency stays approximately constant. Pruning drastically speeds up execution.

Figure 7: Comparison of Normal Pruning (—■—) and SomeGather (—●—) on Accuracy Versus Communication Trade-off. SomeGather's accuracy remains approximately constant with decreasing communication, whereas normal pruning's accuracy degrades significantly.

0 % to 90 % accelerates the attention block by 3.08× at 256 features and 4.37× at 512 features and the residual block by 3.21× and 4.68×, respectively. Overall, pruned inference latency scales from central execution to 16 devices by factors of 3.8× (256 features) and 10.96× (512 features) for the attention block and 1.73× and 6.45× for the residual block.

CATS thus parallelizes computation efficiently. Together with SomeGather's pruning, CATS delivers scalable acceleration despite communication latency.

Remark 1. *The reported latencies depend strongly on the underlying hardware: faster radios and slower processors increase speedups and vice versa. Nevertheless, the fundamental behavior of* CATS *and SomeGather remain unchanged.*

5.4 Q3 – Accuracy of SomeGather

Figure 7 compares SomeGather with normal pruning in terms of prediction error versus communication savings. For normal pruning, prediction error rises steadily as communication is reduced, whereas SomeGather maintains an almost constant error across the entire range. Thus, the gains reported in Sections 5.2 and 5.3 are achieved without sacrificing accuracy.

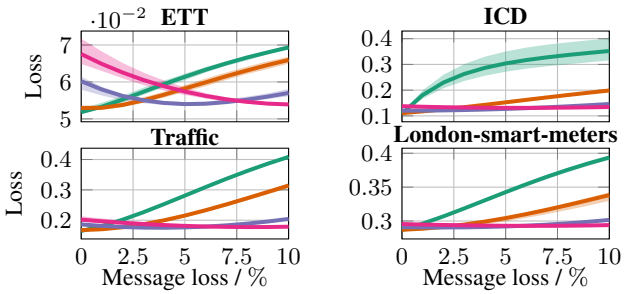

Figure 8: Test Loss of Models Versus Message Loss. We train transformer models with 0 % (—■—), 1 % (—■—), 5 % (—■—) and 10 % (—■—) MD probability. Models trained without MD are highly sensitive to message loss, whereas models trained with MD are robust, performing best near the message loss probabilities they were trained for.

5.5 Q4 – Robustness Against Message Loss

Figure 8 analyzes MD against message loss. Without MD, test accuracy degrades rapidly as message loss increases, whereas MD-trained transformers are markedly more robust and perform best when training and deployment message-loss probabilities match. They still fall short of the original, loss-free network: test losses increase by about 4.2 % for ETT, 23.6 % for ICD, 8.5 % for Traffic and 2.5 % for London-smart-meters, which we attribute to unrecoverable loss in the final layers. Nonetheless, MD substantially improves inference-time robustness to message loss.

6 Conclusions

In this work, we presented CATS, a framework that enables DTI on ultra-low-power wireless devices in mesh networks. We co-designed partitioning, communication and training mechanisms that reduce per-device RAM, flash, and compute while remaining communication-efficient. This allows multiple devices to collaboratively execute transformer models several times larger than what any single device can sustain, as shown in our experiments. Our results further show that pruning inside SomeGather yields substantial RAM, communication and latency savings while maintaining accuracy, and that MD significantly improves robustness to packet losses in wireless networks. However, these gains come with some trade-offs that motivate future work.

First, as each attention head is currently mapped to a single device, the number of heads upper-bounds the number of supported devices. Future work could distribute single heads across multiple devices, to further improve scalability. Second, our pruning and dropout mechanisms are trained for a fixed deployment configuration (number of devices, expected message loss), which is less practical for expensive foundation models reused across diverse settings. An important direction for future research is to equip pre-trained transformers with CATS-like communication awareness without retraining for each network configuration.

Taken together, our results show that DTI is feasible "beyond the edge" on ultra-low-power wireless devices with tight resource budgets, paving the way for more capable systems built from highly constrained devices.

Acknowledgements

This work was supported by the German Research Foundation (DFG) within the priority program 1914 (grant TR 1433/2) and within the Emmy Noether project NextIoT (ZI 1635/2-1), and by the LOEWE initiative (Hesse, Germany) within the emergenCITY center (LOEWE/1/12/519/03/05.001(0016)/72).

The authors gratefully acknowledge the computing time provided to them at the NHR Center NHR4CES at RWTH Aachen University (p0021919).

We thank Andrés Posada Moreno, Lukas Wildberger, Paul Brunzema, Paul Kruse and Fabian Mager for insightful discussions.

References

[Abdi *et al.*, 2020] Afshin Abdi, Saeed Rashidi, Faramarz Fekri, and Tushar Krishna. Restructuring, pruning, and adjustment of deep models for parallel distributed inference. *arXiv preprint arXiv:2008.08289*, 2020.

[Baumann *et al.*, 2020] Dominik Baumann, Fabian Mager, Ulf Wetzker, Lothar Thiele, Marco Zimmerling, and Sebastian Trimpe. Wireless control for smart manufacturing: Recent approaches and open challenges. *Proceedings of the IEEE*, 2020.

[Bochem *et al.*, 2025] Severin Bochem, Victor JB Jung, Arpan Suravi Prasad, Francesco Conti, and Luca Benini. Distributed inference with minimal off-chip traffic for transformers on low-power MCUs. In *Design, Automation & Test in Europe Conference (DATE)*. IEEE, 2025.

[Borges *et al.*, 2014] Luis M Borges, Fernando J Velez, and António S Lebres. Survey on the characterization and classification of wireless sensor network applications. *IEEE Communications Surveys & Tutorials*, 2014.

[Chai *et al.*, 2024] Yuan Chai, Xiao-Jun Zeng, and Zixu Liu. The future of wireless mesh network in next-generation communication: A perspective overview. *Evolving Systems*, 2024.

[Cheng *et al.*, 2024] Yujun Cheng, Zhewei Zhang, and Shengjin Wang. RCIF: Towards robust distributed DNN collaborative inference under highly lossy networks. In *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2024.

[Disabato *et al.*, 2021] Simone Disabato, Manuel Roveri, and Cesare Alippi. Distributed deep convolutional neural networks for the Internet of Things. *IEEE Transactions on Computers*, 2021.

[Dosovitskiy *et al.*, 2021] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

[Du *et al.*, 2024] Jiangsu Du, Yuanxin Wei, Shengyuan Ye, Jiazhi Jiang, Xu Chen, Dan Huang, and Yutong Lu. Codesigning transformer architectures for distributed inference with low communication. *IEEE Transactions on Parallel and Distributed Systems*, 2024.

[Ferrari *et al.*, 2011] Federico Ferrari, Marco Zimmerling, Lothar Thiele, and Olga Saukh. Efficient network flooding and time synchronization with Glossy. In *Proceedings of the 10th ACM/IEEE International Conference on Information Processing in Sensor Networks*, 2011.

[Godahewa *et al.*, 2021] Rakshitha Godahewa, Christoph Bergmeir, Geoffrey I Webb, Rob J Hyndman, and Pablo Montero-Manso. Monash time series forecasting archive. *arXiv preprint arXiv:2105.06643*, 2021.

[Gräfe *et al.*, 2026] Alexander Gräfe, Fabian Mager, Marco Zimmerling, and Sebastian Trimpe. RockNet: Distributed learning on ultra-low-power devices. *ACM Transactions on Cyber-Physical Systems*, 2026.

[Hao *et al.*, 2025] Yuntao Hao, Nan Ding, Weiguo Xia, Hongwei Ge, and Li Xu. DNN partitioning for cooperative inference in edge intelligence: Modeling, solutions, toolchains. *ACM Computing Surveys*, 2025.

[Herrmann *et al.*, 2018] Carsten Herrmann, Fabian Mager, and Marco Zimmerling. Mixer: Efficient many-to-all broadcast in dynamic wireless mesh networks. In *16th ACM Conference on Embedded Networked Sensor Systems*. ACM, 2018.

[Ho *et al.*, 2006] Tracey Ho, Muriel Médard, Ralf Koetter, David R. Karger, Michelle Effros, Jun Shi, and Ben Leong. A random linear network coding approach to multicast. *IEEE Transactions on Information Theory*, 2006.

[Hou and Ohtsuki, 2025] Zhangcheng Hou and Tomoaki Ohtsuki. Loss-adapter: Addressing network packet loss in distributed inference for lossy IoT environments. *IEEE Internet of Things Journal*, 2025.

[Hu and Li, 2024] Chenghao Hu and Baochun Li. When the edge meets transformers: Distributed inference with transformer models. In *44th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2024.

[Itahara *et al.*, 2022] Sohei Itahara, Takayuki Nishio, Yusuke Koda, and Koji Yamamoto. Communication-oriented model fine-tuning for packet-loss resilient distributed inference under highly lossy IoT networks. *IEEE Access*, 2022.

[Jamshed *et al.*, 2022] Muhammad Ali Jamshed, Kamran Ali, Qammer H Abbasi, Muhammad Ali Imran, and Masood Ur-Rehman. Challenges, applications, and future of wireless sensors in Internet of Things: A review. *IEEE Sensors Journal*, 2022.

[Jian *et al.*, 2023] Tong Jian, Debashri Roy, Batool Salehi, Nasim Soltani, Kaushik Chowdhury, and Stratis Ioannidis. Communication-aware DNN pruning. In *IEEE Conference on Computer Communications*, 2023.

[Jung and Lee, 2023] Sehun Jung and Hyang-Won Lee. Optimization framework for splitting DNN inference jobs over computing networks. *Computer Networks*, 2023.

[Kaur and Jadhav, 2023] Ishmeet Kaur and Adwaita Janardhan Jadhav. Survey on computer vision techniques for

Internet of Things devices. In *International Conference on Industry 4.0, Artificial Intelligence, and Communications Technology (IAICT)*. IEEE, 2023.

[Kingma and Ba, 2015] Diederik P. Kingma and Jimmy Ba. ADAM: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[Lai *et al.*, 2018] Liangzhen Lai, Naveen Suda, and Vikas Chandra. CMSIS-NN: Efficient neural network kernels for ARM Cortex-M CPUs. *arXiv preprint arXiv:1801.06601*, 2018.

[Leentvaar and Flint, 1976] Krijn Leentvaar and Jan Flint. The capture effect in FM receivers. *IEEE Transactions on Communications*, 1976.

[Liu *et al.*, 2025a] Xiang Liu, Yijun Song, Xia Li, Yifei Sun, Huiying Lan, Zemin Liu, Linshan Jiang, and Jialin Li. Efficient partitioning vision transformer on edge devices for distributed inference. In *45th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2025.

[Liu *et al.*, 2025b] Xiao Liu, Lijun Zhang, Deepak Ganesan, and Hui Guan. Communication-efficient multi-device inference acceleration for transformer models. *arXiv preprint arXiv:2505.19342*, 2025.

[Mao *et al.*, 2017] Jiachen Mao, Xiang Chen, Kent W Nixon, Christopher Krieger, and Yiran Chen. MoDNN: Local distributed mobile computing system for deep neural network. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2017.

[Nie *et al.*, 2023] Yuqi Nie, Nam H Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. A time series is worth 64 words: Long-term forecasting with transformers. In *The 11th International Conference on Learning Representations*, 2023.

[Prasad *et al.*, 2024] Arpan Suravi Prasad, Moritz Scherer, Francesco Conti, Davide Rossi, Alfio Di Mauro, Manuel Eggimann, Jorge Tomás Gómez, Ziyun Li, Syed Shakib Sarwar, Zhao Wang, et al. Siracusa: A 16 nm heterogenous RISC-V SoC for extended reality with at-MRAM neural engine. *IEEE Journal of Solid-State Circuits*, 2024.

[Qin *et al.*, 2023] Minghai Qin, Chao Sun, Jaco Hofmann, and Dejan Vucinic. Disco: Distributed inference with sparse communications. *arXiv preprint arXiv:2302.11180*, 2023.

[Rahaman and Azharuddin, 2022] Md Mohinur Rahaman and Md Azharuddin. Wireless sensor networks in agriculture through machine learning: A survey. *Computers and Electronics in Agriculture*, 2022.

[Samikwa *et al.*, 2023] Eric Samikwa, Antonio Di Maio, and Torsten Braun. DISNET: Distributed micro-split deep learning in heterogeneous dynamic IoT. *IEEE Internet of Things Journal*, 2023.

[Sanislav *et al.*, 2021] Teodora Sanislav, George Dan Mois, Sherali Zeadally, and Silviu Corneliu Folea. Energy harvesting techniques for Internet of Things (IoT). *IEEE Access*, 2021.

[Sofi *et al.*, 2022] A Sofi, J Jane Regita, Bhagyesh Rane, and Hieng Ho Lau. Structural health monitoring using wireless smart sensor network – an overview. *Mechanical Systems and Signal Processing*, 2022.

[Srinivasan *et al.*, 2010] Kannan Srinivasan, Prabal Dutta, Arsalan Tavakoli, and Philip Levis. An empirical study of low-power wireless. *ACM Transactions on Sensor Networks (TOSN)*, 2010.

[Stahl *et al.*, 2021] Rafael Stahl, Alexander Hoffman, Daniel Mueller-Gritschneder, Andreas Gerstlauer, and Ulf Schlichtmann. DeeperThings: Fully distributed CNN inference on resource-constrained edge devices. *International Journal of Parallel Programming*, 2021.

[Von Birgelen *et al.*, 2018] Alexander Von Birgelen, Davide Buratti, Jens Mager, and Oliver Niggemann. Self-organizing maps for anomaly localization and predictive maintenance in cyber-physical production systems. *Procedia CIRP*, 2018.

[Wei *et al.*, 2024] Yuanxin Wei, Shengyuan Ye, Jiazhi Jiang, Xu Chen, Dan Huang, Jiangsu Du, and Yutong Lu. Communication-efficient model parallelism for distributed in-situ transformer inference. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2024.

[Wen *et al.*, 2025] Dong Wen, Guanping Liang, Tianyun Li, Lin Chen, Junnan Li, and Tao Li. EasyViT: An adaptive collaborative edge computing framework for vision transformer. *IEEE Internet of Things Journal*, 2025.

[Xu *et al.*, 2023] Guanyu Xu, Zhiwei Hao, Yong Luo, Han Hu, Jianping An, and Shiwen Mao. DeViT: Decomposing vision transformers for collaborative inference in edge devices. *IEEE Transactions on Mobile Computing*, 2023.

[Zhang *et al.*, 2025a] Kai Zhang, Hengtao He, Shenghui Song, Jun Zhang, and Khaled B Letaief. Communication-efficient distributed on-device LLM inference over wireless networks. *arXiv preprint arXiv:2503.14882*, 2025.

[Zhang *et al.*, 2025b] Kai Zhang, Qinmin Yang, Chao Li, Xin Sun, and Jiming Chen. Missing data recovery methods on multivariate time series in IoT: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 2025.

[Zhou *et al.*, 2021] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021.

[Zimmerling *et al.*, 2020] Marco Zimmerling, Luca Mottola, and Silvia Santini. Synchronous transmissions in low-power wireless: A survey of communication protocols and network services. *ACM Computing Surveys*, 2020.

[Zong *et al.*, 2025] Mingyu Zong, Arvin Hekmati, Michael Guastalla, Yiyi Li, and Bhaskar Krishnamachari. Integrating large language models with Internet of Things: Applications. *Discover Internet of Things*, 2025.