Recent Developments in Transformer Inference Deployment on FPGA Platforms: A Survey

Preprint — 2026

Arjan Blankestijn
Computer Architecture for Embedded Systems, University of Twente
Enschede, The Netherlands
a.blankestijn-2@student.utwente.nl

Uraz Odyurt
Faculty of Engineering Technology, University of Twente
Enschede, The Netherlands
u.odyurt@utwente.nl

Amirreza Yousefzadeh
Computer Architecture for Embedded Systems, University of Twente
Enschede, The Netherlands
a.yousefzadeh@utwente.nl

Abstract

With the rapid and continuous growth in the incorporation of machine learning models based on the Transformer architecture, capable deployment is in high demand. In this context, capable deployment refers to operational performance aspects, e.g., throughput and latency, as well as efficiency aspects, e.g., energy consumption. When it comes to the task of inference using such models, purpose-built hardware accelerators provide a lucrative alternative to common deployment choices, such as Central Processing Units (CPUs) and Graphics Processing Units (GPUs). The Field Programmable Gate Array (FPGA) platforms category is an example of such alternative accelerators, promising implementation flexibility, energy efficiency, improved latency and suitability for on-site deployment. We investigate the most recent advances, trends, and design choices for Transformer inference on FPGA platforms. We perform a systematic literature review, extracting and delving into preferred techniques for implementation and optimisation. This study and the provided taxonomy of topics could act as a guide for researchers from the academia and industry alike.

Keywords

Transformer, FPGA, Hardware accelerator, Compression, Model synthesis

1 Introduction

Machine Learning (ML) has dramatically reshaped the landscape of computational algorithm design in numerous contexts. The inherent data parallelism present in ML models has proven to be highly advantageous. Additionally, specialised hardware accelerator architectures have been built and incorporated, increasing the computational performance of ML models even further. Well-known examples of such specialised hardware architectures beyond Central Processing Units (CPUs) are, Graphics Processing Units (GPUs), Neural Processing Units (NPUs), Field-Programmable Gate Arrays (FPGAs), and Neuromorphic computing. Some of these designs pre-existed and found to be effective ML accelerators for different reasons, i.e., inherent computational compatibilities or fine-grained customisability. Two main examples are GPUs and FPGAs.

Amongst these hardware accelerator architectures, FPGAs stand out by portraying unique characteristics. FPGAs have attracted substantial attention as a viable architecture for accelerating deep learning workloads. Their reconfigurability, together with the capacity for custom, hardware-level optimisations, makes FPGAs particularly attractive for ML inference. Example use-cases benefiting from hardware-level optimisations are requirements for low-latency operation, high-throughput operation, or any combination with desired balance between latency and throughput. FPGA-based platforms present a compelling alternative to conventional CPUs and GPU accelerators, especially where power footprint and physical size requirements impose limitations, or where on-site deployment and installation is required.

While established deep learning ML model architectures, e.g., Convolutional Neural Networks (CNNs), have been taking advantage of hardware accelerators, particularly FPGAs, there is an emerging architecture, the Transformer. The Transformer model architecture [1] has gained prominence due to its powerful ability to handle distant and obscure dependencies and to efficiently parallelise computations. Originally conceived for tasks such as machine translation, the Transformer architecture has evolved into a fundamental component for many high-performance models. However, the computational load and memory demands associated with Transformer inference remain significant, with much higher demands compared to the established model architectures. These challenges are especially apparent when Transformer models are intended for real-time applications and deployed in resource-constrained environments.

In short, as opposed to older, more established ML model architectures, the methodology and tooling for deployment of Transformers on FPGAs remain incomplete. We strive to capture the current state-of-the-art for the context of *Transformer inference running on FPGA platforms*. As such, answers to a number of questions are to be sought after:

- *What are the performance indicators?*
- *How to improve these performance indicators?*
- *Which implementation or resource management aspects are to be leveraged to achieve better results?*
- *Which optimisations are considered? Which tooling is available and what are the limitations of such tooling?*

Scope of the survey. We provide a systematic literature review, aiming to consolidate and critically evaluate existing research on Transformer model inference on FPGAs. It must be emphasised that the focus is *inference* and not training. As this particular angle is rather active, we focus on the more recent research from the last 2 years, 2024–May 2025.

Following this introduction, the background and motivation, covering alternative surveys and fundamental topics, is provided in Section 2. The survey methodology itself is given in Section 3, followed by the extracted taxonomy of topics in Section 4. Details on storage, tooling, model compression, and IP design from the covered articles are provided in Sections 5 to 8, respectively. Analysis of performance metrics and trends is elaborated in Section 9, alongside concluding remarks in Section 10.

2 Background and motivation

In recent years, growing amounts of research effort has focused on adapting the Transformer model designs to leverage the advantages provided by the FPGA technology. Researchers have explored various optimisation strategies, ranging from quantization and pruning to custom memory management and parallel processing frameworks, in order to bridge the gap between the theoretical performance of Transformers and the practical limitations posed by FPGA hardware. Furthermore, as plotted in Figure 1, the number of articles on this topic has risen significantly in the last 2 years, with 153 and 51 articles for 2024 and 2025 till the end of May, respectively. This sharp increase in the number of published articles reflects the increased momentum in channelled research effort.

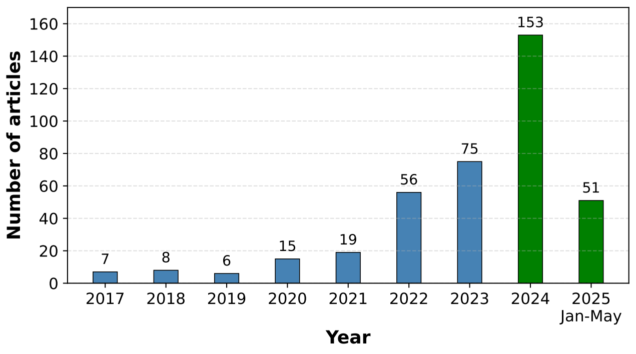

Figure 1: Distribution of article counts per year (till the end of May 2025), focusing on the topic of Transformer inference on FPGA's and considering the three repositories: IEEE Xplore, Scopus, and arXiv as sources. The scope for this survey is the last two bars (in green).

As such, now can be considered as yet another interesting point in time to observe emerging trends, consider reached conclusions, and identify stated future work approaches. Surveys on the topic have been conducted in the past, with one year frequencies, but not covering the 2024–2025 span. Notably, in works by Kang *et al.* [2] and Chitty-Venkata *et al.* [3], authors have surveyed the topic extensively. Other, use-case specific surveys exist as well, focusing on the applications on Transformers in specific areas, such as Vision Transformers (ViT) [4] and Large Language Models (LLMs) [5]. A detailed coverage of these surveys is listed in Table 1.

As the field is extremely active and rapidly evolving, we have opted to focus on the latest developments and achievements. Accordingly, this survey targets all FPGA-based Transformer inference articles published during the year 2024 and the year 2025 till the end of May, using a clear and reproducible selection methodology,

Table 1: Previous surveys and their coverage as a point of comparison to this survey. Note that last listed year, usually the publication year, is never fully covered. These surveys do not specify the exact collection period.

Year(s)	Platform(s)	Use-case	Systematic	Survey
–2023	ASIC, FPGA, GPU, CPU	Non-specific	No	[3]
–2024	ASIC, FPGA	Non-specific	No	[2]
2025	ASIC, FPGA, GPU	ViT	Yes	[4]
–2024	ASIC, FPGA, GPU, CPU	LLM	No	[5]

i.e., systematic. Our survey does not focus on specific use-case categories.

2.1 Transformer architecture

The Transformer architecture, introduced by Vaswani *et al.* [1], is built entirely on attention mechanisms and is widely used for sequence processing use-cases, e.g., natural language processing and computer vision. At a high level and in its original form, a Transformer consists of an encoder–decoder structure. Many modern applications of the Transformer architecture make use of the encoder block, e.g., BERT, or the decoder block, e.g., GPT-style models, without the other.

Each encoder layer comprises a *multi-head self-attention* segment, followed by a position-wise feed-forward network. The self-attention mechanism allows every token in the input sequence to attend to all other tokens, enabling the model to capture both short- and long-range dependencies. For an input matrix $\mathbf{X}$, three learned linear projections produce the *Query*, *Key*, and *Value* matrices:

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_Q, \quad \mathbf{K} = \mathbf{X}\mathbf{W}_K, \quad \mathbf{V} = \mathbf{X}\mathbf{W}_V,$$

where $\mathbf{W}_Q$, $\mathbf{W}_K$, and $\mathbf{W}_V$ are the corresponding weight matrices. Accordingly, the attention output is computed as

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{SoftMax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V},$$

where d_k is the dimensionality of the key vectors. Through multi-head attention, this computation is replicated across multiple heads, enabling the model to capture diverse relational patterns.

Each decoder layer adds an additional *masked self-attention* segment to the design, to preserve autoregressive behaviour. A cross-attention segment that attends to the encoder outputs is included as well. Both encoder and decoder layers make use of residual connections and layer normalisation [6] to ensure stable gradients and improve training efficiency.

The attention mechanism is permutation-invariant, for which Transformers rely on *position embeddings* to inject order information into it. These can be either fixed sinusoidal embeddings or learned embeddings. The architectural design eliminates the need for recurrence or convolution, allowing for highly parallel computation. Figure 2 (inspired by [1]) depicts the standard Transformer architecture with all the aforementioned segments.

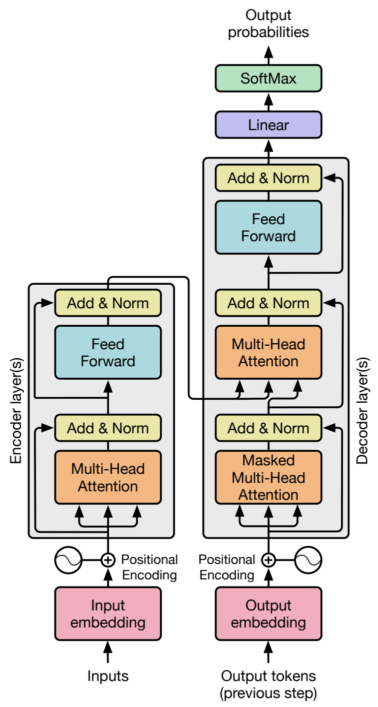

Figure 2: The original Transformer model architecture with encoder and decoder segments.

2.2 FPGA technology

Field-Programmable Gate Arrays (FPGAs) are reconfigurable hardware platforms that allow designers to implement custom digital circuits, tailored to specific computational workloads [7]. Unlike fixed-function Application-Specific Integrated Circuits (ASICs) or general-purpose CPUs, an FPGA consists of an array of programmable logic blocks. Blocks such as Look-Up Tables (LUTs), Flip-Flops (FF), and interconnect resources can be reconfigured after manufacturing. This reconfigurability makes FPGAs attractive to implement purpose-built accelerators, e.g., for accelerating emerging ML architectures, such as Transformers, where computational patterns evolve rapidly [8].

Modern FPGAs also include a set of hardened resources, providing high-performance arithmetic and memory capabilities. These typically include Digital Signal Processing (DSP) blocks for efficient multiply-accumulate operations, Block RAMs (BRAMs) and UltraRAMs (URAMs) for low-latency on-chip storage, and high-bandwidth interfaces for connecting to off-chip Dynamic Random-Access Memory (DRAM) [9, 10]. By tailoring computation to these resources, designers can exploit massive fine-grained parallelism and reduce the memory bottlenecks commonly encountered in deep learning workloads [11].

A key advantage of FPGAs is the ability to define bespoke dataflows and memory hierarchies. Designers can structure computation around streaming architectures, systolic arrays, or pipelined parallel units, depending on the characteristics of the target algorithm [12]. On-chip buffers are frequently used to minimise expensive off-chip memory accesses, improving latency. Data reuse strategies, such as tiling, caching, and weight buffering, play a central role in achieving high throughput [13].

FPGA development flows typically rely on Hardware Description Languages (HDLs) such as Verilog or VHDL. However, High-Level Synthesis (HLS) tools have become increasingly common, allowing hardware to be generated from C/C++-like specifications or domain-specific toolchains [14, 15]. This has significantly reduced development time and made FPGA-based acceleration more accessible in ML research and industry. This is due to higher level abstractions, i.e., lower implementation complexities, provided by HLS.

Overall, FPGAs offer a flexible and energy-efficient platform for accelerating compute- and memory-intensive tasks. Their customisability, combined with support for high degrees of parallelism, makes them particularly well suited for implementing Transformer components such as attention mechanisms, matrix multiplications, and hierarchical buffering schemes [16]. Needless to say, there is a strong element of use-case specificity involved, making implementation of reusable multi-use designs a constant challenge.

3 Survey method & pruning protocols

The literature considered for this review has been gathered and selected in a systematic and semi-automatic fashion. the processing and pruning of collected query responses is performed using Python scripts developed by the authors. The code is available online[1]. An overview diagram of the steps with article counts resulting from each step is given in Figure 3.

Repositories. All covered articles are collected from three source repositories, IEEE Xplore[2], Scopus[3], and arXiv[4]. The collection from arXiv is performed dynamically through the available Application Programming Interface (API)[5].

Query. A standard query, made up of highly relevant terms was formulated, reflecting our focused topics. The query,

> *"transformer AND fpga AND (inference OR ai OR acceleration)"*,

was used for searching within all three repositories, resulting in a total of 411 articles.

Pruning - Duplicate removal. Since multiple repositories are considered and especially in the presence of preprint versions from arXiv, pruning for duplicates is necessary. This pruning step resulted in 335 articles remaining.

Pruning - Date filter (< 2024). A date filter applied to bring the focus to publications from 2024 onwards, till the end of May 2025. This pruning step resulted in 161 articles remaining.

Pruning - Abstract focus. Subsequently, all articles are filtered based on whether or not the keyword "transformer" is present in the abstract. This pruning step is to ensure the inclusion of articles

[1]https://virtualdetector.com/trackcore-f/repository.html
[2]https://ieeexplore.ieee.org/Xplore/home.jsp
[3]https://www.scopus.com/
[4]https://arxiv.org/
[5]https://info.arxiv.org/help/api/index.html

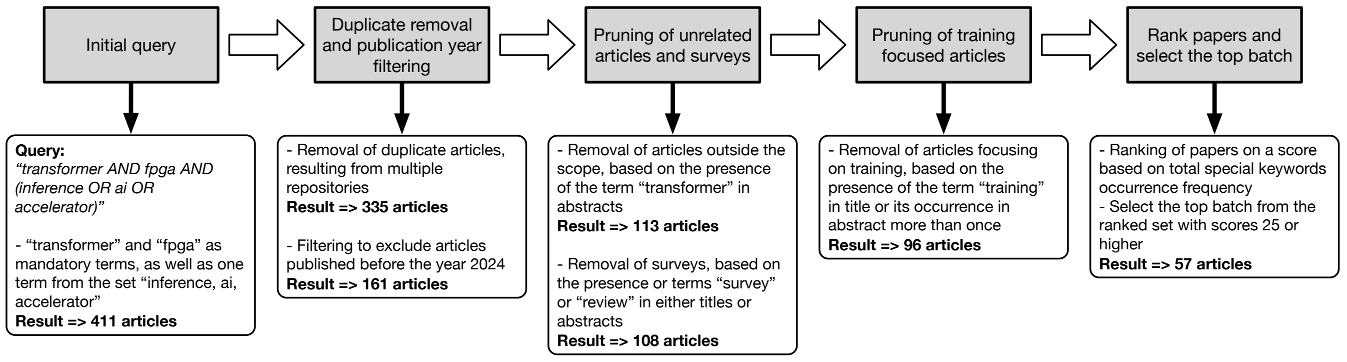
Initial query
Duplicate removal and publication year filtering
Pruning of unrelated articles and surveys
Pruning of training focused articles
Rank papers and select the top batch
Query: "transformer AND fpga AND (inference OR ai OR accelerator)"
- "transformer" and "fpga" as mandatory terms, as well as one term from the set "inference, ai, accelerator"
Result => 411 articles
- Removal of duplicate articles, resulting from multiple repositories
Result => 335 articles
- Filtering to exclude articles published before the year 2024
Result => 161 articles
- Removal of articles outside the scope, based on the presence of the term "transformer" in abstracts
Result => 113 articles
- Removal of surveys, based on the presence or terms "survey" or "review" in either titles or abstracts
Result => 108 articles
- Removal of articles focusing on training, based on the presence of the term "training" in title or its occurrence in abstract more than once
Result => 96 articles
- Ranking of papers on a score based on total special keywords occurrence frequency
- Select the top batch from the ranked set with scores 25 or higher
Result => 57 articles

Figure 3: Article collection and pruning workflow, with details on pruning protocols and resulting article counts after each step.

with Transformer models as the primary focus. It resulted in 113 articles remaining.

Pruning - Survey removal. The response to the query naturally includes surveys relevant to the topic, which are to be set aside. Articles with terms "survey" or "review" in either the title or the abstract are considered surveys and filtered. This pruning step resulted in 108 articles remaining.

Pruning - Articles on training. Finally, as our aim is to cover inference, articles with a primary focus on training are to be pruned. This is done based on whether the keyword "training" is included in the title or if it occurs more than once in the abstract. The remaining articles after this pruning step is 96.

Ranking. Collection of articles during any survey will result any a spectrum of articles in terms of relevance. To limit the amount of articles investigated for this review to the ones with the highest relevancy, a ranking is applied to the collection. Points are awarded to each article based on the frequency of keywords, selected from sub-topics relevant to our topic, within the title and the abstract. Table 2 covers the list of keywords per sub-topic category. The articles are ranked based on the number of points and with 25 as the threshold score. The top 57 articles, achieving 25 points or higher, are considered for this survey. This collection is our primary source. The most relevant of these 57 articles are processed by the authors.

4 Taxonomy of Transformer inference on FPGAs

Given the collected articles and based on our review of these, we provide a taxonomy of approaches and techniques covered by these literature while deploying Transformer models on FGPA platforms for inference, drawn in Figure 4.

The taxonomy provides a natural hierarchy, encompassing the two main aspects addressed in the literature, *Implementation* and *Optimisation.* We shall use this hierarchy to organise the remainder of this survey. The taxonomy is derived from the analysis of recurring themes and design choices from the collected articles. The authors have incorporated their knowledge of similar implementations to arrive at a more meaningful structure. While not exhaustive, Figure 4 reflects the dominant implementation and optimisation techniques reported and serves as a conceptual guide for positioning individual contributions. The performed ranking of articles based on relevancy to the topic elevates the taxonomy higher in its representativeness.

Table 2: Keyword groups for ranking articles with scores based on occurrence frequency.

Category	Keywords
Synthesis	HLS, high level synthesis, high-level synthesis, register transfer level
Memory	memory, storage, on-chip, off-chip
Compression	compression, quantization, pruning, sparsity, sparse
Acceleration	acceleration, accelerator, reconfigurable devices, systolic, fpga
AI terms	ai, artificial intelligence, inference
Misc. terms	weight, weights, computation, efficient, constrain

5 Model weights storage

Memory transfer can be a big bottleneck in inference latency. Therefore, the way that the model weights and biases are stored and handled can have a big impact on performance. Various techniques exist for storing the weights and biases of a model. The two most popular approaches are storing of all weights on-chip or dynamically loading of parameters from off-chip memory, as needed.

Ideally, all parameters are permanently stored on-chip, i.e., on the FPGA, in order to minimise memory transfer. However, this is not always feasible when dealing with large models and/or smaller FPGAs. Additionally, this approach can be impractical for generic accelerators aimed at supporting multiple types of models. As such, a rather common technique is to store the weights in external memory or have the weights loaded dynamically trough the host CPU. Note that most hardware platforms provide a CPU, often time an ARM core, alongside the FPGA, hence the host CPU.

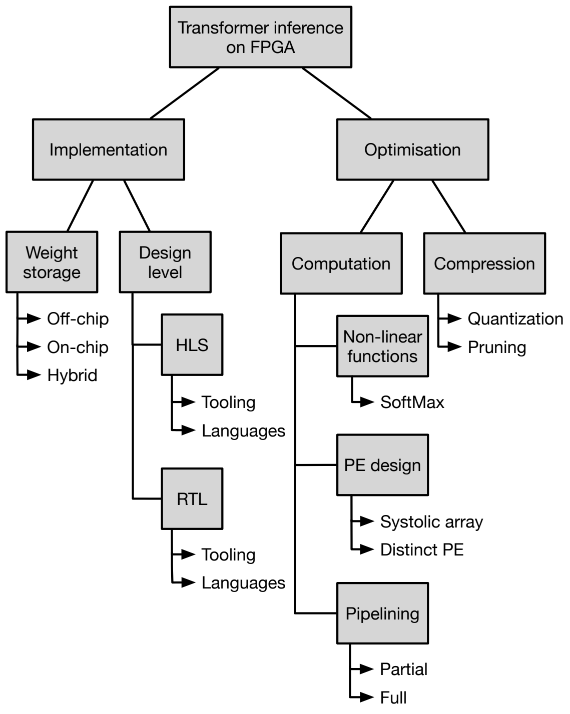

Figure 4: Extracted taxonomy of prevalent approaches for Transformer inference on FPGA platforms, summarising the main subjects, design dimensions, and optimisation techniques identified in the surveyed literature.

5.1 Off-chip storage

Accelerators making use of off-chip memory access are generally capable of supporting many different models and model sizes, since these are not limited by the available on-chip memory. In one such approach, Kabir *et al.* [17] propose a Transformer accelerator that is runtime-adaptive and makes use of off-chip memory access. The architecture makes use of separate *Load Units* for inputs, weights, and biases. For instance, on-chip weight buffers for the attention computation are 2-dimensional arrays, where the array size depend on the tiling size. For each iteration when handling each tile, the partial data is loaded from external memory. Since biases are generally smaller in size, these are stored in registers. The biases are loaded into registers while the attention module is computing the **Q**, **K**, and **V** matrices. Afterwards, the biases are then added to the matrices. FAMOUS [18] is another accelerator from the same authors that employs a very similar system. The weight matrices are loaded in on-chip memory for every iteration and the biases are transferred to internal registers during the matrix multiplication step.

Optimisations to reduce the number of memory transfers can still be made, even when making use of off-chip memory. For example, Li and Chen [19] propose a *Tiled Matrix Multiplication Accelerator* for self-attention in Transformers. This accelerator focuses only on the **Q**, **K**, and **V** projections. For each matrix multiplication, the **Q**, **K**, and **V** weight matrices are transferred to the FPGA. However, in order to minimise memory transfers, the input matrix is only transferred to the FPGA once for the **Q** projection. Then, the input matrix is stored in an on-chip buffer for **K** and **V** projections.

5.2 On-chip storage

The most performance-effective way to store a model's parameters is to store these on-chip. On-chip storage minimises off-chip memory access, which in turn has a big impact on overall performance. As mentioned, considering limited on-chip memory, storing all weights and activations on-chip is not always possible. Another advantage of permanently storing weights using on-chip storage is to facilitate the use of extensive pipelining, as exemplified by Guo *et al.* with HG-PIPE [20]. In order to take advantage of pipelining, the authors of HG-PIPE propose an implementation that stores as many weights and activations on-chip as possible.

Alternatively, instead of having all weights fixed in on-chip memory, it is also possible to store these in on-chip buffers with the possibility of loading in different weights for different models. Take ME-ViT for instance, authored by Marino [21], which is an accelerator for Vision Transformers. This accelerator will load in all parameters from off-chip memory and store these in on-chip buffers before starting the inference. Parameters that have to be re-used multiple times during a single inference are strategically buffered in on-chip memory. Then, the Memory Efficient Processing Element (ME-PE) is able to perform in 3 different modes: Linear Projection, Multi-Headed Self-Attention and Multi-Layer Perceptron. Any intermediate results during inference are also kept in on-chip buffers. This architecture ensures that there are minimal amount of memory transfers between off-chip and on-chip variants. This implementation does suffer from on-chip memory limitations. It must be noted that this architecture does allow for inference of multiple different models, compared to storing of parameters permanently on-chip.

Yet another example of an accelerator storing all the model weights and activations in on-chip memory is ProTEA, authored by Kabir *et al.* [22]. The authors propose an efficient tiling strategy, allowing large models to be stored in on-chip memory. ProTEA is runtime configurable, enabling inference of different models. All the model weights are loaded into on-chip memory ahead of the inference.

5.3 Performance comparison

Table 3 presents a comparison in resource consumption between articles employing off-chip memory access and articles storing model weights on-chip during inference. It is important to note that a considerable number of articles do not make it clear if the incorporated accelerator loads in all weights from off-chip memory into on-chip memory prior to inference, or if the implementation accesses off-chip memory during inference. We would qualify the former approach as "on-chip memory storage" as there would be no off-chip memory access during the inference itself. Therefore, it is likely that a significant portion of implementations that we have categorised as "off-chip" are actually qualify as "on-chip". It is difficult to make precise comparisons here. Nevertheless, it is interesting to see if a trend can be noticed.

As it can be seen in Table 3, accelerators making use of off-chip memory access generally have a higher resource consumption. With significantly more FF, LUTs, DSPs and BRAM usage. At the same time, accelerators keeping all the weights in on-chip memory have a significantly higher throughput and much lower latency. This may seem illogical, as one would expect that implementations with higher resource consumption should result in better performance. However, this can be explained by the memory access time required due to the incorporation of off-chip memory, which is a considerable bottleneck, severely limiting both throughput and latency.

We can conclude from Table 3 that in general, utilising as much persistent on-chip storage as possible results in much lower inference latencies and higher throughput. However, if the goal is to make an accelerator suitable for a wider variety of models, then the only possible architecture could be one where the weights are not stored on-chip, but rather off-chip.

6 Incorporated tooling

Various techniques are available to develop solutions for FPGAs. These techniques might vary depending on the model or manufacturer of the considered FPGA. We can categorise the available tooling and programming languages under two main existing design levels, Register-Transfer Level (RTL) and High-Level Synthesis (HLS). HLS tools simplify development substantially, as there is no requirement for knowledge of low-level RTL-design and hardware description languages. The opposite is the case with RTL tooling, with a focus on low-level hardware design, which also provides the opportunity for ad hoc optimisations. Considering these two design level, i.e., design methodologies, the prevalence of tooling from each is investigated below, alongside the performance and resource consumption impact analysis of selected tooling.

Table 4 lists how often each type of tooling is used within the analysed pool of articles in this survey. The table is divided into two categories: RTL and HLS. The RTL category includes Verilog, SystemVerilog, and VHDL, while the HLS category includes Vitis, Versel ACAP, hls4ml, and Unknown tooling. The Unknown category is for articles not specifying the utilised tooling.

Unfortunately, not all articles specify the incorporated tooling. For the articles that do specify it, the distribution between the usage of RTL tooling and HLS tooling is rather evenly split, with 12 articles making use of RTL tooling and 13 utilising HLS tooling. Within RTL tooling, the Hardware Description Language (HDL) Verilog[6] is by far the most popular choice. Only 2 articles used other tooling, namely [31] and [48] using SystemVerilog and VHDL, respectively. A similar trend can be seen within HLS tooling, where Vitis HLS is by far the most popular choice. Other tooling used includes Versel ACAP (by [36]) and hls4ml (by [49]).

6.1 Frameworks

A small selection of articles set out to develop or extended frameworks for the development of Transformer accelerators. Jiang *et al.* [49] extended the popular HLS tool, hls4ml [56, 57] to include support for any Tensorflow-based Transformer model. hls4ml is an open-source Python package that can translate machine learning models from frameworks like Keras and PyTorch into HLS code for deployment on FPGAs. The authors of [49] have used Vivado in the implementation of the Transformer for hls4ml.

On the RTL side, Ling *et al.* [48] developed VHDL templates which enable developers to translate existing models onto an FPGA accelerator, without a deep understanding of FPGA development workflow. In order to generate an accelerator, Python scripts translate these models using VHDL templates, resulting in VHDL files. This approach ensures a seamless translation of trained and quantized models into FPGA accelerators. The authors note that in the future they aim to support mixed-precision quantization and to improve the energy efficiency of resulting accelerators.

6.2 HLS vs RTL

Table 5 provides a comparison of the average resource usage, as well as performance metrics, i.e., throughput in Giga Operations Per Second (GOPS) and power in Watts. The comparison is between articles using RTL design level tooling with the ones utilising HLS design level tooling. It is important to consider that the architecture and design of the accelerator at hand has a much bigger impact on these figures. However, these figures can portray an overall trend. For instance, articles making use of HLS tooling, on average seem to have much more FF and LUT consumption. Additionally, the power consumption is on average twice that of articles making use of RTL tooling. This shows that in general, using RTL tooling results in more efficient designs. Interestingly, the average throughput is rather higher for HLS tooling designs. One possible explanation for this is that HLS tooling allows for more time and effort to be spent on efficient computations, i.e., allowing the tooling to convert the model design to the best possible RTL design.

7 Model compression

Various model compression techniques exist in order to reduce model size, enabling efficient usage of on-chip storage instead of off-chip storage. Furthermore, certain pruning techniques can reduce the model size and complexity, which in turn can result in increased inference performance. Popular methods include *quantization* and *pruning*. The following covers and compares commonly used quantization and pruning techniques. This includes comparing different bit-widths and if the same quantization levels can be used across different types of parameters and layers. The impact on latency, throughput and accuracy of compression techniques must be taken into account as well.

7.1 Quantization

Using 8-bit quantization is by far the most common practice in articles using quantization, 13 out of 30. Some papers use higher bit-widths, e.g., 12 or 16 bits ([52] and [26, 38], respectively), in order to retain a higher accuracy. On the contrary, other implementations opt for even lower bit-widths in order to optimise and reduce computational latency even further. Another approach is to make use of mixed quantization, e.g., [28, 31]. In this approach, important parameters are quantized with a higher bit-width, whereas parameters that have less influence on the final accuracy are quantized with a lower bit-width.

[6]Also standardised as IEEE 1364: https://standards.ieee.org/ieee/1364/2052/

Table 3: Resource utilisation and performance comparison between storage approaches, for off-chip and on-chip. 6 articles with unknown storage approach are not covered here. Note that not every field is provided in every article. Missing cells are designated with "n/a".

FF	LUT	DSP	BRAM	Platform	Throughput (GOPS)	Power (W)	Latency (ms)	Article
				Storage approach: Off-chip				
114K	128K	768	n/a	ZCU102	1 023.3	23.48	n/a	[23]
n/a	77K	147	n/a	VCK190	n/a	n/a	n/a	[24]
n/a	n/a	1 024	n/a	ZCU102	780	7.43	n/a	[25]
1 507K	557K	5 096	1 581	XCU200	2 750	28.23	0.08	[26]
n/a	271K	1 863	609.5	Alveo U50	301.9	14.35	n/a	[27]
n/a	391K	3 612	n/a	n/a	40	11.9	n/a	[17]
943K	574K	6 345	1 253	Alveo U280	n/a	n/a	n/a	[28]
n/a	n/a	n/a	n/a	Alveo U55C	n/a	n/a	n/a	[29]
n/a	n/a	n/a	n/a	n/a	n/a	9	17.51	[30]
704K	993K	3 612	n/a	VCK190	44	n/a	165	[22]
222K	115K	1 248	n/a	KV260	n/a	10	50	[31]
143K	123K	1 850	458	ZCU102	97.04	11.5	25.76	[32]
2 446K	1 593K	10 848	1 746	AMD V80	n/a	n/a	n/a	[33]
n/a	n/a	n/a	n/a	ZCU102	385.5	9.86	n/a	[34]
162K	122K	78	691.5	ZCU102	3 894.74	8.68	n/a	[35]
n/a	n/a	n/a	n/a	VCK190	n/a	47.5	56	[36]
n/a	493K	5 016	n/a	Alveo U280	1 223.9	n/a	14.57	[37]
n/a	n/a	3 482	1 162	Alveo U50	1 420	48	n/a	[38]
578K	472K	9 024	1 520	VCU128	8 204	43.2	n/a	[39]
661K	1 284K	4 157	3 148	Alveo U55C	184	n/a	0.597	[18]
103K	71K	1 050	126	KV260	2.85	n/a	110	[19]
682 500	506 055.56	3 842	1 200.15	–	1 453.66	20.075	49.9177	Total average
				Storage approach: On-chip				
n/a	n/a	n/a	n/a	Alveo U200	n/a	31.8	n/a	[21]
73K	110K	0	177	ZCU102	944.87	1.39	n/a	[40]
n/a	669K	312	1 006	VCK190	17 795	46.7	n/a	[20]
291K	161K	1945	812	ZCU102	n/a	16.71	1.77	[41]
548K	274K	2520	912	ZCU102	1 387.59	7.95	n/a	[42]
139K	118K	2147	283	XCZU9EG	2 330.2	21.37	13.98	[43]
1 083K	544K	10 812	1 903	Alveo U250	986.3	n/a	1.17	[44]
426 800	312 666.67	2 956	848.83	–	4 688.79	20.987	5.64	Total average

HG-PIPE, by Guo *et al.* [20], makes extensive use of LUT-based MAC units, i.e., multiply–accumulate operators implemented using FPGA lookup tables instead of dedicated DSP blocks. If the operands are quantized to 3 bits, the multiplication operation can be decomposed into six boolean functions, each consuming 6 bits. Only 6 LUT-6s are required for such an operation. This technique reduces the number of required DSPs from more than 10 000 to only 744. Furthermore, reducing the bit-width from 4 to 3 only reduces the final accuracy from 73.30% to 69.60%.

Jiang *et al.* in [49] have extended the hls4ml framework [57] to support the Transformer model. hls4ml allows for varying precision across different layers. The authors have tested various fixed precision configurations, including both Post Training Quantization (PTQ) and Quantization Aware Training (QAT) across three different Transformer models. The authors settled on using 6 bits for two of these models and 10 bits for the last model. Using any lower bit-widths would result in a significant drop in the Area Under the Receiver Operating Characteristics curve (AUC). This shows that the optimum quantization bit-width can heavily depend on the model under evaluation.

Du *et al.* [40] propose an accelerator for binary Transformers. In this case, all model parameters are 1-bit quantized. To be precise, all weighs are binarised to {-1, 1}, activations of ReLU and SoftMax are binarised to {0, 1}, and all other activations are binarised to {-1, 1}. The resulting architecture uses significantly fewer resources and provides higher throughput compared to other sparsity-aware accelerators. The major downside to this approach is that it is not feasible to quantize a model to 1-bit during post-training quantization.

Xiang *et al.* [51] make use of an Activation-aware Weight Quantization (AWQ) technique, in order to significantly reduce the memory footprint. This technique is based on the fact that only 1% of

Table 4: Comparison of synthesis tooling used in different article, with a significant number lacking elaboration.

Design level	Language/Tool	Count	Articles
RTL	Verilog	10	[45, 46, 23, 25, 40, 41, 34, 37, 39, 47]
	SystemVerilog	1	[31]
	VHDL	1	[48]
HLS	Vivado/Vitis HLS	11	[21, 24, 17, 29, 30, 22, 32, 35, 38, 18, 19]
	Versal ACAP	1	[36]
	hls4ml	1	[49]
	Unknown	2	[20, 50]
Unknown	Unknown	11	[51, 26, 28, 42, 33, 43, 44, 52, 53, 54, 55]

the model weights may have a significant impact on model performance. Thus, preserving these weights can maintain accuracy while reducing the memory footprint. The technique works by performing scaling on a per-channel basis, based on the activation distribution. This way, the entire matrix can be quantized to very lower precisions (INT4, INT3) while maintaining model accuracy. Evaluation of this quantization technique results in a 55.10% reduction in memory footprint with only a 2.82 percentage points reduction in model accuracy compared to a baseline.

HEAT by Zhao *et al.* [23] is a Transformer Accelerator employing a hybrid-precision quantization scheme. The resulting model weights are quantized to an average of 5.71 bits, with only a 0.526% accuracy loss for the model. This technique works by splitting the weight matrices based on a predefined threshold. Values within this threshold are considered "normal" and values outside of the threshold are considered outliers. The normal values are quantized to 5 bits while the outliers are quantized to 8 bits. Choosing a suitable threshold value is important. In prior work the threshold is set to a constant value. However, in [23], the threshold is determined by a line search based on a constant outlier ratio of 3.5%.

Byun *et al.* use a Hessian-driven quantization approach [60] in [41]. The authors note that using such an approach has challenges, such as the fact that quantized weights using a parameter-wise approach result in irregular precision patterns. This is not ideal for hardware accelerators. To overcome such limitations, the authors propose a row-wise approach. This technique works by splitting the matrices into important and unimportant rows. The aggregate parameter sensitivity is determined for each row within every matrix of each layer. The rows are sorted based on their sensitivity. Then, all weights are quantized with lower precision while quantizing only the top 1% of weight rows with a higher precision. This process is repeated iteratively until the desired level of accuracy is achieved. The architecture of the accelerator is designed to be able to efficiently handle both the low- and the high-precision calculations. The resulting accelerator increases energy efficiency up to 14.57 times compared to existing accelerators.

Another method for mixed precision quantization is implemented by Li *et al.* [31]. This article implements a Transformer model for the task of license plate recognition. In order to minimise memory overhead, the authors propose a mixed precision quantization method, optimising the allocation of low bit-width for each layer. Additionally, a Dynamic Precision Adaption (DPA) algorithm is proposed to optimise the allocation between the integer and fractional part. This algorithm considers the magnitude of input and allocates the appropriate bit-width for the largest input value in order to avoid overflow. The resulting algorithm increases the model weight compression by 2.44 times, as opposed to 1.95 times compression when using 8-bit fixed point quantization, while maintaining identical model accuracy.

7.2 Model pruning

Li *et al.* [46] note that the output of the inner product of the **Q** and **K** matrices can contain up to 95% zeroes or close to zero values. These terms have little contribution to the overall model accuracy. Eliminating the computation of these products can significantly reduce the amount of computation, while not affecting the model accuracy by much. The sparsity cannot be predicted before the actual matrix computation though. However, the authors propose a dynamic sparse computation method. First, a low-precision 4-bit computation between the **Q** and **K** matrices is performed. This results in a mask matrix, indicating positions of non-zero values. This mask is used to select the **K** matrix during full 16-bit computation. Compared to other accelerators, this implementation can result in improved throughput and energy efficiency.

Wang [45] proposes a block-wise balanced pruning technique for model compression. The technique works by first splitting up the parameter matrix into smaller blocks of predetermined sizes. Then, pruning is performed for each block, such that each block has the same number of parameters removed. Pruning is done by ranking parameters in descending order. The smallest P parameters are removed depending on the pruning ratio, P. The second step is to use an efficient memory storage pattern that makes use of a binary bitmap, indicating non-zero values. Finally, the actual non-zero values are stored in a row-major order. This pruning technique maintains a model accuracy of 97.70% while reaching an 87.00% pruning ratio.

For FNM-Trans, by Zhang *et al.* [26], authors propose an accelerator that makes use of full $N : M$ Sparsity. $N : M$ sparsity is a structured sparsity technique where there are N non-zero elements for every M elements. FNM-Trans does apply $N : M$ sparsity to both the attention mechanism and weights. The authors propose a pruning algorithm to obtain high sparsity without impacting final model accuracy by a significant amount. The algorithm works in two stages. The first stage starts with $N = M$ for both attention and weights. Then, N_a and N_w are reduced with 1, followed by a model retraining using the new N_a and N_w parameters. This process of slowly reducing N and subsequent trainings is continued until the model accuracy drops too much. The second stage is very similar to the first stage, except that instead of reducing both N_a and N_w at the same time, first N_a is reduced until the minimum N_a is reached. The same is done with N_w. The resulting model has up to 93.75% attention sparsity and up to 75% weight sparsity with only a 3.43% reduction in model accuracy.

Table 5: Average resource usage and performance metrics, covering throughput (in GOPS) and power (in Watts), compared between RTL vs HLS design level tooling. Note that not every field is provided in every article. Missing cells are designated with "n/a".

FF	LUT	DSP	BRAM	Platform	Throughput (GOPS)	Power (W)	Article
Tooling category: RTL							
169K	125K	1 032	n/a	ZCU102	695.37	4.78	[45]
152K	208K	832	n/a	ZCU102	634.27	3.72	[46]
114K	128K	768	n/a	ZCU102	1 023.3	23.48	[23]
n/a	n/a	1 024	n/a	ZCU102	780	7.43	[25]
73K	110K	0	177	ZCU102	944.87	1.39	[40]
291K	161K	1 945	812	ZCU102	n/a	16.71	[41]
222K	115K	1 248	n/a	KV260	n/a	10	[31]
n/a	1 008K	9 091	1 819	Alveo U250	n/a	n/a	[58]
n/a	n/a	n/a	n/a	ZCU102	385.5	9.86	[34]
n/a	493K	5 016	n/a	Alveo U280	1 223.9	n/a	[37]
578K	472K	9 024	1 520	VCU128	8 204	43.2	[39]
n/a	n/a	n/a	n/a	XC7S15	n/a	0.075	[48]
228 428	313 555	2 998	1 082	–	2 093	13	Total average
Tooling category: HLS							
n/a	n/a	n/a	n/a	Alveo U200	n/a	31.8	[21]
n/a	77K	147	n/a	VCK190	n/a	n/a	[24]
n/a	669K	312	1 006	VCK190	17 795	46.7	[20]
n/a	391K	3 612	n/a	n/a	40	11.9	[17]
n/a	n/a	n/a	n/a	Alveo U55C	n/a	n/a	[29]
n/a	n/a	n/a	n/a	n/a	n/a	9	[30]
704K	993K	3 612	n/a	Alveo U55C	44	n/a	[22]
143K	123K	1 850	458	ZCU102	97.04	11.5	[32]
162K	122K	78	691.5	ZCU102	3 894.74	8.68	[35]
n/a	n/a	n/a	n/a	VCK190	n/a	47.5	[36]
n/a	n/a	3 482	1 162	Alveo U50	1 420	48	[38]
607K	833K	6 225	1 440	VU9P	n/a	n/a	[59]
661K	1 284K	4 157	3 148	Alveo U55C	184	n/a	[18]
n/a	n/a	n/a	n/a	n/a	n/a	n/a	[49]
103K	71K	1 050	126	KV260	2.85	n/a	[19]
396 666	507 000	2 452	1 147	–	2 934	26.885	Total average

8 IP design optimisations

Different computation and architecture designs that can be used for Transformer inference. Often, some form of Processing Elements (PE) are considered. Some implementations include designing dedicated PEs for specific calculations, whereas others employ generic PEs used for various purposes. Beyond PE designs, techniques such as data reuse or pipelining may be considered to increase throughput. Finally, various ways of increasing performance for computing non-linear functions, such as the SoftMax function, has been opted for.

8.1 Computation: SoftMax and non-linear functions

The SoftMax function has a significant performance impact on Transformer inference in FPGAs, due to its reliance on expensive operations. These operations, e.g., exponentiation and division, are highly inefficient on FPGA hardware. The frequent use of such operations in attention layers leads to an increase in computational and memory requirements, while its sequential nature limits parallelism and optimisation options.

A popular method of implementing the SoftMax function is through the use of Look-Up Tables (LUTs). For instance, Li [46] has implemented such a SoftMax computation. However, the author notes that for a 16-bit SoftMax computation, the look-up table will be too large and will not fit onto the available memory. A more efficient SoftMax computation can be achieved by assuming a predefined threshold value and finding the maximum attention score. Then, for all other attention scores, if the difference with the maximum score is bigger than the threshold, it is discarded. In this manner, a minimum valid score is recorded and all valid entries between 1 and the threshold can be stored in a table. Each valid score is subtracted by the minimum value and is used in a look-up table.

The authors of SWAT, Bai *et al.* [29], note that standard implementations of Transformer models involve 3 distinct sequential steps: **QK** multiplication, SoftMax, and **SV** multiplication. Often, each step is broken down into smaller tile-wise operations. However, since the SoftMax function is row-wise data dependent, it is not possible to compute it in a tiling window. This results in high number of memory transfers for loading and storing tile-wise intermediate results. Instead, the authors propose a kernel fusion optimisation technique. The SoftMax function is divided into two separate components: a numerator that does not depend on other elements of the row, and the denominator that depends on the sum of the exponential of all elements of the same row. In this fashion, the denominator can be placed after the **SV** multiplication step, allowing the fusion of all three steps into a "unified row-wise kernel". The restructuring of the SoftMax function [29] is noted in mathematical terms in Equation (1).

$$\begin{aligned} Z_{i,j} &= \sum_{n=0}^{H} S'_{i,n} V_{n,j} = \sum_{n=0}^{H} \frac{\exp(S_{i,n})}{\sum_{l=0}^{H} \exp(S_{i,l})} V_{n,j} \\ &= \frac{1}{\sum_{l=0}^{H} \exp(S_{i,l})} \sum_{n=0}^{H} \exp(S_{i,n}) V_{n,j} \qquad (1) \end{aligned}$$

A similar approach is followed in SALTS by Chen *et al.* [34], a flexible self-attention accelerator with Long Token Support. The authors note that the original SoftMax operation "requires traversing the input vector three times: element-wise exponent, reduced summation, and element-wise division". By re-arranging the SoftMax computation, a fully pipelined design can be realised. Qin *et al.* [38] have also employed the same technique.

8.2 Architecture designs: PE design

A large majority of FPGA accelerator implementations employ *Processing Elements (PEs)* as fundamental building blocks in their design. A PE is a design module dedicated to a specific computation or set of operations. Most works adopt architectures composed of multiple PEs, but differ in how computation is distributed amongst them. A common approach is the use of *systolic arrays* consisting of many small, identical PEs, where each PE performs a simple, local operation and data flows regularly between neighbouring elements. Other implementations organise their architecture around multiple larger, functionally distinct PEs, with each PE responsible for a discrete computation stage or kernel. These architectural strategies are not necessarily mutually exclusive, and several designs combine elements of both to balance parallelism, resource utilisation, and design complexity. Furthermore, it is not always clearly distinguishable from the descriptions in the article which architectural elements are used in a given implementation. Based on the articles surveyed in this work, at least 25 employ PEs arranged in a systolic array, while at least 14 use architectures composed of several distinct PEs. Both collections are listed in Table 6.

Table 6: List of articles taking advantage of different PE implementation approaches, which are not necessarily mutually exclusive.

PE approach	Count	Articles
Systolic arrays	25	[46, 21, 51, 23, 25, 40, 26, 41, 52, 17, 28, 22, 31, 32, 33, 43, 34, 35, 36, 38, 50, 39, 55, 18, 44]
Distinct PEs	14	[45, 46, 23, 26, 22, 31, 32, 33, 34, 36, 38, 39, 18, 44]

An example of an implementation making use of different PE types is FAMOUS [18]. The architecture consist of 3 different modules, each with its own unique PE type. The `QKV` module is responsible for calculating the query, key and value matrices. The next module, called the `QK` module, is responsible for the matrix multiplication between the **Q** and **K** matrices and applying the SoftMax function. Finally, the `SV` module is responsible for multiplication with the Value matrix.

For FlightLLM by Zeng *et al.* [28], a multi-core design has been employed, where each core includes a Matrix Processing Engine (MPE). Each MPE consists of multiple Matrix Processing Units that perform matrix calculations. The MPE supports two separate modes, an MM mode for Matrix-Matrix multiplication and an MV mode for Matrix-Vector multiplication.

8.3 Architecture designs: Pipelining

Pipelining in FPGA design is an approach where operations are divided into separate stages, with each stage performing part of the computation. These stages are connected in sequence, each operating in parallel on different data during every clock cycle. Such a design increases throughput, as new data can enter the pipeline before the preceding data has finished processing, allowing for more efficient use of hardware resources. While in some architectures only a specific computation or part of the inference is pipelined [22], other architectures are designed with full pipelining in mind [20].

ProTEA [22] is an example of an architecture that does not necessarily implement a fully pipelined architecture, but does make use of pipelining within single computation steps. ProTEA has an architecture consisting of different PEs, targeting specific computations. The number of PEs depends on the unrolling factor and initiation interval of the pipelined loop.

An instance of a fully pipelined architecture is HG-PIPE [20]. The architecture consist of a series of block levels, where each represents a single Encoder layer. Pipelining is performed both at the inter-block level and within each block. HG-PIPE employs a hybrid grained pipeline approach, i.e., while some computations can occur simultaneously, other computations are bound to start once previous steps are complete. A diagram describing the execution impact of different pipelining paradigms is provided in Figure 5 (inspired by [20]). The incorporated elements in addition to PE are: General Matrix-Matrix Multiplication (GeMM); Parallel-In Parallel-Out (PIPO), which is a type high-bandwidth parallel data register; Auxiliary (Aux) as supporting or helper hardware modules, e.g., non-linear functions; and First-In First-Out (FIFO), which is used for ordered buffering to achieve data streaming between accelerator stages.

Overall, a large majority of articles mention the use of pipelining in some form, either within single computation steps or on a larger architectural scale. In total, 68% of the collected articles explicitly

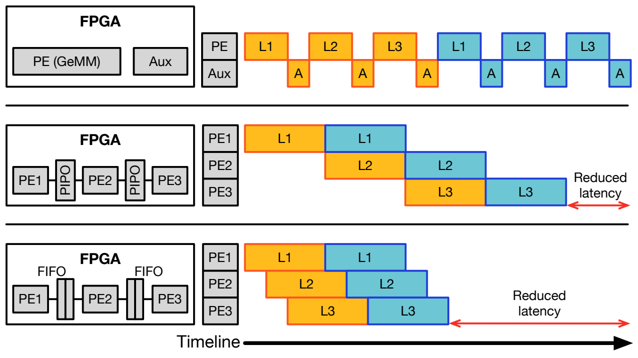

Figure 5: Three pipelining paradigms and resulting effects on the execution timeline, i.e., reduction in latency. From top to bottom: Temporal design, i.e., no pipelining; Coarse-grained pipelining; Fine-grained pipelining.

mention the use of pipelining. The remaining 32% do not make any comments regarding pipelining.

8.4 Architecture designs: Instruction Set Architectures

For this design, FlightLLM [28] and FlightVGM [33] are two accelerators that make use of a custom Instruction Set Architecture (ISA) between the host and the hardware accelerator. The ISA consists of 6 instructions. First, the Load (LD) and Store (ST) instructions covered, which transfer data between off-chip memory and on-chip buffers. The Matrix-Matrix multiplication (MM) and Matrix-Vector multiplication (MV) instructions are added for matrix multiplication operations. The MISC instruction is included for Layer Normalisation and SoftMax calculations. Finally, the SYS instruction covers the synchronisation between the host CPU and multiple Super Logic Regions (SLRs), after each inference execution is completed.

9 Insights and discussion

Given what we have gathered from the pool of articles in this survey, there are comparisons and analyses worthy of sharing. When it comes to performance comparison, the two relatively common metrics to rely on are: *Throughput in GOPS* and *Energy efficiency in GOPS/Watts.*

9.1 Performance comparison: Throughput

Focusing on throughput as a performance metric, Table 7 lists the top 10 implementations with the highest throughput in Giga Operations Per Second (GOPS), sorted from high to low. GOPS is used to measure how many billion operations can be performed per second. In the context of Transformer machine learning inference on FPGAs, GOPS gives an idea of the raw compute throughput achieved when running inference workloads. As can be seen in Table 7, HG-PIPE [20] has by far the highest throughput, with a throughput more than twice as much as the second best implementation by Liu *et al.* [39].

HG-PIPE also has the highest number of consumed LUTs, which is expected as the implementation relies on LUT-based computations. Interestingly, while Table 3 indicates that implementations utilising on-chip memory have on average a much higher throughput, which is also a loose theoretical expectation, this does not hold. Considering the top 10 highest throughput implementations, only 4 take advantage of on-chip storage, proving that it is still possible to achieve relatively equal performance when using off-chip memory. However, we must note that the top contender, HG-PIPE, is based on on-chip storage and it achieves a considerably higher throughput compared to the 2nd best implementation.

9.2 Performance comparison: Energy efficiency

Deducing energy efficiency is less straightforward as it depends on, simply put, the amount of work being done, which translates to data throughput per platform at hand. When comparing implementations, it is tempting to focus only on raw performance metrics, such as throughput and power. However, these metrics both are highly dependent on the implemented model itself, as well as the size of the FPGA, i.e., both are use-case and experiment specific. Therefore, energy efficiency (GOPS/Watts), i.e., the amount of energy spent per unit of throughput, can be an additional metric to consider for a fair comparison. Similar to our throughput metric listing, Table 8 lists the top 10 implementations with the best energy efficiency, sorted from high to low. This selection is limited to those articles reporting both power usage and throughput. As it can be seen, the top two implementations both use binarised quantization, which allows for very fast and efficient computations. However, switching to binarised weights is not always possible for all models. Furthermore, a considerable number of implementations (5 articles, half) within this top 10 batch, use RTL-level design instead of HLS, much higher than the average use of RTL amongst all articles. According to Table 4, the average use of RTL is about 30%. Although, these ratios are to be taken with a grain of salt, since a considerable number of articles, seen in Table 4, do not specify the choice of design level.

9.3 Effective implementations

Figure 6 plots throughput versus energy efficiency for FPGA-based inference accelerators listed in Tables 7 and 8. Designs with both metrics available are shown with red markers, while designs reporting only throughput are shown with blue markers. Note that not every implementation provides both metrics and there is an intersection between the two tables. The dashed line denotes the Pareto frontier, representing designs for which no other design achieves both higher throughput and higher energy efficiency, simultaneously. The plot highlights the trade-off between raw performance and energy efficiency and as we can see, a small number of Pareto-optimal designs best balance these competing objectives.

As an alternative perspective, Figure 7 shows how energy efficiency evolves as we move from the highest-throughput designs to the lower-throughput ones. Here, throughput-sorted designs are given with throughput and energy efficiency plotted on separate axes. While throughput decreases monotonically by construction, energy efficiency exhibits non-monotonic behaviour, indicating

Table 7: Top 10 implementations with the highest throughput in GOPS, sorted from high to low. Note that not every field is provided in every article. Missing cells are designated with “n/a”.

Design level	Storage	FF	LUT	DSP	BRAM	Bit-width	Innovation	Throughput (GOPS)	Article
HLS	On-chip	n/a	669K	312	1 006	3	LUT based MAC	**17 795**	[20]
RTL	Off-chip	578K	472K	9 024	1 520	n/a	Flexible floating point	**8 204**	[39]
HLS	Off-chip	162K	122K	78	691.5	1	Binary weights	**3 894.74**	[35]
n/a	Off-chip	1507K	557K	5 096	1 581	16	$N:M$ sparsity	**2 750**	[26]
n/a	On-chip	139K	118K	2 147	283	8	Weight loop dataflow	**2 330.2**	[43]
HLS	Off-chip	n/a	n/a	3 482	1 162	16	Attention fusion	**1 420**	[38]
n/a	On-chip	548K	274K	2 520	912	1	Binary weights	**1 387.59**	[42]
RTL	Off-chip	n/a	493K	5 016	n/a	8, 4	Sorting engine	**1 223.9**	[37]
RTL	Off-chip	114K	128K	768	n/a	5	Hybrid quantization	**1 023.3**	[23]
n/a	On-chip	1083K	544K	10 812	1 903	8	Redundancy aware	**986.3**	[44]

Table 8: Top 10 implementations with the highest power efficiency (throughput/power), sorted from high to low. Note that not every field is provided in every article. Missing cells are designated with “n/a”.

Design level	FF	LUT	DSP	BRAM	Bit-width	Innovation	Efficiency (GOPS/W)	Article
RTL	73K	110K	0	177	1	Binary weights	**679.76** (944.87 / 1.39)	[40]
HLS	162K	122K	78	691.5	1	Binary weights	**448.70** (3 894.74 / 8.68)	[35]
HLS	n/a	669K	312	1 006	3	LUT based MAC	**381.05** (17 795 / 46.7)	[20]
RTL	578K	472K	9 024	1 520	n/a	Flexible floating point	**189.91** (8 204 / 43.2)	[39]
n/a	548K	274K	2 520	912	1	Binary weights	**174.54** (1 387.59 / 7.95)	[42]
n/a	139K	118K	2 147	283	8	Weight loop dataflow	**109.04** (2 330.2 / 21.37)	[43]
RTL	n/a	n/a	1 024	n/a	8	Intra-/inter-layer fusions	**104.98** (780 / 7.43)	[25]
n/a	1 507K	557K	5 096	1 581	16	$N:M$ sparsity	**97.41** (2 750 / 28.23)	[26]
RTL	114K	128K	768	n/a	5	Hybrid quantization	**43.58** (1 023.3 / 23.48)	[23]
RTL	n/a	n/a	n/a	n/a	8	n/a	**39.10** (385.5 / 9.86)	[34]

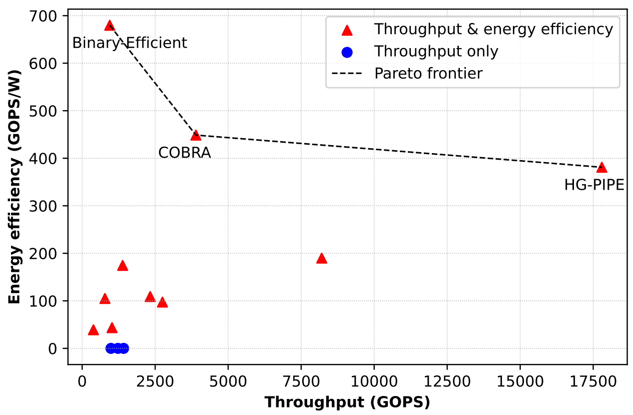

Figure 6: Throughput versus energy efficiency for FPGA-based inference accelerators, designating the Pareto-optimal designs and the Pareto frontier.

that high throughput does not necessarily imply poor energy efficiency. This further motivates the use of a Pareto-based analysis.

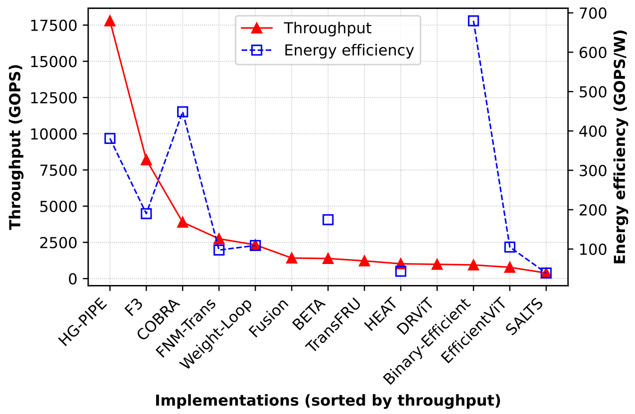

Figure 7: Energy efficiency trends for FPGA-based inference accelerators, with respect to ranked throughput.

Legend. HG-PIPE [20], F3 [39], COBRA [35], FNM-Trans [26], Weight-Loop [43], Fusion [38], BETA [42], TransFRU [37], HEAT [23], DRViT [44], Binary-Efficient [40], EfficientViT [25], SALTS [34].

Note that while such plots are revealing, it is hard to arrive at any solid conclusion. As little generality can be claimed beyond applied methodologies for implementations targeting FPGAs, such analyses are to be taken with a grain of salt. After all, a tremendous amount of customisation goes into each individual implementation and these custom techniques are not common amongst the implementations.

9.4 Standardised model benchmarking

One of the major challenges in conducting surveys on this topic is the difficulty of making meaningful comparisons between different accelerator implementations. The primary reason is the inconsistent choice of Transformer models, or different variants of the same model, to be evaluated, making fair comparisons largely infeasible. The chosen model has a significant impact on performance, resource utilisation, and energy efficiency of accelerator implementations.

Several models are frequently used in the literature, such as BERT [61], Swin [62], and DeiT [63]. However, these models represent families of architectures rather than fixed specifications and exist in numerous configurations, differing in structure and parameter counts. Consequently, reported performance figures often reflect model complexity just as much as architectural or algorithmic innovation.

Using one or more standard Transformer benchmark models would make future work easier to compare by standardising the model structure and workload. In addition, a common benchmarking approach with common sets of performance metrics, measurement conditions, and reporting practices, will further reduce uncertainty and improve reproducibility. It is important to ensure that reported performance gains are primarily resulting from accelerator design choices, rather than differences in model variants or evaluation methods. While standard models and benchmarking practices would not replace application-specific evaluations, they could serve as a common reference point. At a minimum, consistently reporting the evaluated model's size, configuration, parameter count, and benchmarking setup, will provide valuable context and enable more meaningful comparisons.

10 Conclusion

We have provided a comprehensive and systematic review of recent articles on the topic of *Transformer model inference deployment on FPGA platforms*. We have followed a repeatable protocol for collection and pruning of articles, which also renders the review easily expandable by addition of new articles. The extracted taxonomy, Figure 4, depicts a high-level, but complete hierarchy of the focus area within the literature. The review culminates in top-10 listings based on the two main performance metrics commonly reported, throughout (GOPS) and power (W), from which we derive power efficiency (GOPS/W) as a more meaningful ranking metric.

We observe that there is no absolute preference across any of the examined categorical dimensions. Whether considering design abstraction level (HLS vs. RTL), memory organisation (off-chip vs. on-chip), pipelining strategies, or quantization schemes, the choices remain rather implementation-specific and application-driven. While this diversity highlights the inherent flexibility of FPGA platforms, it also underscores the difficulty of drawing direct comparisons across published works. In particular, we have touched upon the lack of standardised benchmarking methodologies, as well as incomplete reporting of architectural and implementation details, as a main challenge to comparisons.

Quantization deserves special emphasis as a double-edged sword. While reduced precision and lower bit-width arithmetic can yield substantial gains in performance and energy efficiency, they may also incur non-negligible degradation in model accuracy, to the point that some ML models will no longer be viable if quantized. Addressing this trade-off, alongside improved reporting practices and benchmark standardisation, represents a key opportunity for future work in FPGA-based Transformer inference.

A final and ever-present challenge lies in the trade-off between architectural flexibility and peak performance. Highly specialised FPGA accelerators, tailored to a specific Transformer model or configuration, can achieve substantial gains in throughput and energy efficiency, but at the cost of limited support for alternative models or workloads. In contrast, more flexible designs that accommodate a wider range of model variants and parameters typically incur additional overheads, reducing raw performance. This trade-off is inherent rather than incidental: designs optimised for maximum performance generally sacrifice flexibility, while general-purpose accelerators must accept lower peak efficiency. Consequently, the appropriate balance remains application-dependent and driven by deployment requirements.

Acknowledgments

This publication is part of the project ZORRO with project number KICH1.ST02.21.003 of the research programme Key Enabling Technologies (KIC), which is (partly) financed by the Dutch Research Council (NWO).

References

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*. I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, (Eds.)

[2] Beom Jin Kang, Hae In Lee, Seok Kyu Yoon, Young Chan Kim, Sang Beom Jeong, Seong Jun O, and Hyun Kim. 2024. A survey of FPGA and ASIC designs for transformer inference acceleration and optimization. *Journal of Systems Architecture*. doi:10.1016/j.sysarc.2024.103247.

[3] Krishna Teja Chitty-Venkata, Sparsh Mittal, Murali Emani, Venkatram Vishwanath, and Arun K. Somani. 2023. A survey of techniques for optimizing transformer inference. *Journal of Systems Architecture*. doi:10.1016/j.sysarc.2023.102990.

[4] Shaibal Saha and Lanyu Xu. 2025. Vision Transformers on the Edge: A Comprehensive Survey of Model Compression and Acceleration Strategies. (2025). doi:10.48550/arXiv.2503.02891.

[5] Nikoletta Koilia and Christoforos Kachris. 2024. Hardware Acceleration of LLMs: A comprehensive survey and comparison. (2024). doi:10.48550/arXiv.2409.03384.

[6] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer Normalization. (2016). doi:10.48550/arXiv.1607.06450.

[7] Ian Kuon and Jonathan Rose. 2006. Measuring the gap between FPGAs and ASICs. In *Proceedings of the 2006 ACM/SIGDA 14th International Symposium on Field Programmable Gate Arrays*. doi:10.1145/1117201.1117205.

[8] Stylianos I. Venieris, Alexandros Kouris, and Christos-Savvas Bouganis. 2018. Toolflows for Mapping Convolutional Neural Networks on FPGAs: A Survey and Future Directions. *ACM Comput. Surv.* doi:10.1145/3186332.

[9] AMD Xilinx. 2025. *UltraScale Architecture and Product Data Sheet: Overview (DS890)*. AMD Xilinx. https://docs.amd.com/v/u/en-US/ds890-ultrascale-overview.

[10] Intel. 2024. *Stratix® 10 GX/SX Device Overview*. Intel. https://cdrdv2.intel.com/v1/dl/getContent/670537?fileName=s10-overview-683729-670537.pdf.

[11] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. 2015. Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. doi:10.1145/2684746.2689060.

[12] Andrew Boutros, Aman Arora, and Vaughn Betz. 2025. Field-Programmable Gate Array Architecture for Deep Learning: Survey and Future Directions. *Proceedings of the IEEE*. doi:10.1109/JPROC.2025.3623023.

[13] Yongming Shen, Michael Ferdman, and Peter Milder. 2017. Maximizing CNN Accelerator Efficiency Through Resource Partitioning. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*. doi:10.1145/3079856.3080221.

[14] Andrew Canis, Jongsok Choi, Mark Aldham, Victor Zhang, Ahmed Kammoona, Jason H. Anderson, Stephen Brown, and Tomasz Czajkowski. 2011. LegUp: high-level synthesis for FPGA-based processor/accelerator systems. In *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*. doi:10.1145/1950413.1950423.

[15] Razvan Nane et al. 2016. A Survey and Evaluation of FPGA High-Level Synthesis Tools. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*. doi:10.1109/TCAD.2015.2513673.

[16] Christoforos Kachris. 2025. A Survey on Hardware Accelerators for Large Language Models. *Applied Sciences*. doi:10.3390/app15020586.

[17] Ehsan Kabir, Jason D. Bakos, David Andrews, and Miaoqing Huang. 2024. A Runtime-Adaptive Transformer Neural Network Accelerator on FPGAs. (2024). doi:10.48550/arxiv.2411.18148.

[18] Ehsan Kabir, Md. Arafat Kabir, Austin R. J. Downey, Jason D. Bakos, David Andrews, and Miaoqing Huang. 2024. FAMOUS: Flexible Accelerator for the Attention Mechanism of Transformer on UltraScale+ FPGAs. (2024). doi:10.48550/arXiv.2409.14023.

[19] Richie Li and Sicheng Chen. 2025. Design and Implementation of an FPGA-Based Hardware Accelerator for Transformer. (2025). doi:10.48550/arXiv.2503.16731.

[20] Qingyu Guo, Jiayong Wan, Songqiang Xu, Meng Li, and Yuan Wang. 2024. HG-PIPE: Vision Transformer Acceleration with Hybrid-Grained Pipeline. (2024). doi:10.48550/arXiv.2407.17879.

[21] Kyle Marino, Pengmiao Zhang, and Viktor Prasanna. 2024. Me-vit: a single-load memory-efficient fpga accelerator for vision transformers. (2024). doi:10.48550/arXiv.2402.09709.

[22] Ehsan Kabir, Jason D. Bakos, David Andrews, and Miaoqing Huang. 2024. ProTEA: Programmable Transformer Encoder Acceleration on FPGA. In *SC24-W: Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis*. doi:10.1109/SCW63240.2024.00074.

[23] Pan Zhao, Donghui Xue, Licheng Wu, Liang Chang, Haining Tan, Yinhe Han, and Jun Zhou. 2025. HEAT: Efficient Vision Transformer Accelerator With Hybrid-Precision Quantization. *IEEE Transactions on Circuits and Systems II: Express Briefs*. doi:10.1109/TCSII.2025.3547340.

[24] Linfeng Zhong, Qingyu Guo, Runsheng Wang, Yuan Wang, and Meng Li. 2024. Flexible Yet Efficient Transformer Acceleration with Unified Sparse Attention Support on FPGA. In *2024 IEEE 17th International Conference on Solid-State & Integrated Circuit Technology (ICSICT)*. doi:10.1109/ICSICT62049.2024.10831534.

[25] Haikuo Shao, Huihong Shi, Wendong Mao, and Zhongfeng Wang. 2024. An FPGA-Based Reconfigurable Accelerator for Convolution-Transformer Hybrid EfficientViT. In *2024 IEEE International Symposium on Circuits and Systems (ISCAS)*. doi:10.1109/ISCAS58744.2024.10557992.

[26] Manting Zhang, Jialin Cao, Kejia Shi, Keqing Zhao, Genhao Zhang, Jun Yu, and Kun Wang. 2024. FNM-Trans: Efficient FPGA-based Transformer Architecture with Full N:M Sparsity. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*. doi:10.1145/3649329.3656497.

[27] Zhiyang Liu, Pengyu Yin, and Zhenhua Ren. 2023. An Efficient FPGA-Based Accelerator for Swin Transformer. (2023). doi:10.48550/arXiv.2308.13922.

[28] Shulin Zeng et al. 2024. FlightLLM: Efficient Large Language Model Inference with a Complete Mapping Flow on FPGAs. In *Proceedings of the 2024 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*. doi:10.1145/3626202.3637562.

[29] Zhenyu Bai, Pranav Dangi, Huize Li, and Tulika Mitra. 2024. SWAT: Scalable and Efficient Window Attention-based Transformers Acceleration on FPGAs. (2024). doi:10.48550/arXiv.2405.17025.

[30] Andy He, Darren Key, Mason Bulling, Andrew Chang, Skyler Shapiro, and Everett Lee. 2024. HLSTransform: Energy-Efficient Llama 2 Inference on FPGAs Via High Level Synthesis. (2024). doi:10.48550/arXiv.2405.00738.

[31] Jie Li, Dingjiang Yan, Fangzhou He, Zhicheng Dong, and Mingfei Jiang. 2024. A Mixed-Precision Transformer Accelerator With Vector Tiling Systolic Array for License Plate Recognition in Unconstrained Scenarios. *IEEE Transactions on Intelligent Transportation Systems*. doi:10.1109/TITS.2024.3457815.

[32] Jiale Dong, Wenqi Lou, Zhendong Zheng, Yunji Qin, Lei Gong, Chao Wang, and Xuehai Zhou. 2025. UbiMoE: A Ubiquitous Mixture-of-Experts Vision Transformer Accelerator With Hybrid Computation Pattern on FPGA. In *2025 IEEE International Symposium on Circuits and Systems (ISCAS)*. doi:10.1109/ISCAS56072.2025.11043956.

[33] Jun Liu et al. 2025. FlightVGM: Efficient Video Generation Model Inference with Online Sparsification and Hybrid Precision on FPGAs. In *Proceedings of the 2025 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*. doi:10.1145/3706628.3708864.

[34] Kaiqi Chen, Xinhua Shi, and Jun Han. 2024. SALTS: An Efficient and Flexible Self-Attention Accelerator with Long Token Support on FPGA. In *2024 IEEE 17th International Conference on Solid-State & Integrated Circuit Technology (ICSICT)*. doi:10.1109/ICSICT62049.2024.10831003.

[35] Ye Qiao, Zhiheng Chen, Yian Wang, Yifan Zhang, Yunzhe Deng, and Sitao Huang. 2025. COBRA: Algorithm-Architecture Co-optimized Binary Transformer Accelerator for Edge Inference. (2025). doi:10.48550/arXiv.2504.16269.

[36] Peiyan Dong et al. 2024. EQ-ViT: Algorithm-Hardware Co-Design for End-to-End Acceleration of Real-Time Vision Transformer Inference on Versal ACAP Architecture. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*. doi:10.1109/TCAD.2024.3443692.

[37] Hongji Wang, Yueyin Bai, Jun Yu, and Kun Wang. 2024. TransFRU: Efficient Deployment of Transformers on FPGA with Full Resource Utilization. In *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*. doi:10.1109/ASP-DAC58780.2024.10473976.

[38] Yunji Qin, Wenqi Lou, Chao Wang, Lei Gong, and Xuehai Zhou. 2024. Enhancing Long Sequence Input Processing in FPGA-Based Transformer Accelerators through Attention Fusion. In *Proceedings of the Great Lakes Symposium on VLSI 2024*. doi:10.1145/3649476.3658810.

[39] Zerong He, Xi Jin, and Zhongguang Xu. 2025. F3: An FPGA-Based Transformer Fine-Tuning Accelerator With Flexible Floating Point Format. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*. doi:10.1109/JETCAS.2025.3555970.

[40] Congpeng Du, Seok-Bum Ko, and Hao Zhang. 2024. Energy Efficient FPGA-Based Binary Transformer Accelerator for Edge Devices. In *2024 IEEE International Symposium on Circuits and Systems (ISCAS)*. doi:10.1109/ISCAS58744.2024.10558631.

[41] Woohong Byun, Jongseok Woo, and Saibal Mukhopadhyay. 2024. Hardware-friendly Hessian-driven Row-wise Quantization and FPGA Acceleration for Transformer-based Models. In *Proceedings of the 29th ACM/IEEE International Symposium on Low Power Electronics and Design*. doi:10.1145/3665314.3670806.

[42] Yuhao Ji, Chao Fang, and Zhongfeng Wang. 2024. BETA: Binarized Energy-Efficient Transformer Accelerator at the Edge. In *2024 IEEE International Symposium on Circuits and Systems (ISCAS)*. doi:10.1109/ISCAS58744.2024.10558636.

[43] Yueqi Zhang, Lichen Feng, Hongwei Shan, and Zhangming Zhu. 2024. A 109-GOPs/W FPGA-Based Vision Transformer Accelerator With Weight-Loop Dataflow Featuring Data Reusing and Resource Saving. *IEEE Transactions on Circuits and Systems for Video Technology*. doi:10.1109/TCSVT.2024.3439600.

[44] Xiangfeng Sun, Yuanting Zhang, Qinyu Wang, Xiaofeng Zou, Yujia Liu, Ziqian Zeng, and Huiping Zhuang. 2025. DRViT: A dynamic redundancy-aware vision transformer accelerator via algorithm and architecture co-design on FPGA. *Journal of Parallel and Distributed Computing*. doi:10.1016/j.jpdc.2025.105042.

[45] Saiqun Wang and Hao Zhang. 2024. Efficient FPGA-Based Transformer Accelerator Using In-Block Balanced Pruning. In *2024 13th International Conference on Communications, Circuits and Systems (ICCCAS)*. doi:10.1109/ICCCAS62034.2024.10651591.

[46] Zuohao Li, Yiwan Lai, and Hao Zhang. 2024. Energy Efficient FPGA-Based Accelerator for Dynamic Sparse Transformer. In *2024 13th International Conference on Communications, Circuits and Systems (ICCCAS)*. doi:10.1109/ICCCAS62034.2024.10652850.

[47] Bingyi Zhang, Rajgopal Kannan, Carl Busart, and Viktor K. Prasanna. 2024. VisionAGILE: A Versatile Domain-Specific Accelerator for Computer Vision Tasks. *IEEE Transactions on Parallel and Distributed Systems*. doi:10.1109/TPDS.2024.3466891.

[48] Tianheng Ling, Chao Qian, and Gregor Schiele. 2024. Integer-only Quantized Transformers for Embedded FPGA-based Time-series Forecasting in AIoT. In *2024 IEEE Annual Congress on Artificial Intelligence of Things (AIoT)*. doi:10.1109/AIoT63253.2024.00017.

[49] Zhixing Jiang et al. 2024. Low Latency Transformer Inference on FPGAs for Physics Applications with hls4ml. (2024). doi:10.48550/arXiv.2409.05207.

[50] Mohammad Erfan Sadeghi, Arash Fayyazi, Suhas Somashekar, Armin Abdollahi, and Massoud Pedram. 2024. CHOSEN: Compilation to Hardware Optimization Stack for Efficient Vision Transformer Inference. (2024). doi:10.48550/arXiv.2407.12736.

[51] Maoyang Xiang, Ramesh Fernando, and Bo Wang. 2025. On-Device Qwen2.5: Efficient LLM Inference with Model Compression and Hardware Acceleration. (2025). doi:10.48550/arXiv.2504.17376.

[52] Qiwei Dong, Xiaoru Xie, and Zhongfeng Wang. 2024. SWAT: An Efficient Swin Transformer Accelerator Based on FPGA. In *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*. doi:10.1109/ASP-DAC58780.2024.10473931.

[53] Minseok Seo et al. 2024. IANUS: Integrated Accelerator Based on NPU-PIM Unified Memory System. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*. doi:10.1145/3620666.3651324.

[54] Cheng Xu, Yirong Kan, Renyuan Zhang, and Yasuhiko Nakashima. 2024. An FPGA-Oriented Quantization Approach for Vision Transformer with LUT-Friendly Operations. In *2024 Twelfth International Symposium on Computing and Networking (CANDAR)*. doi:10.1109/CANDAR64496.2024.00042.

[55] Abhishek Moitra, Abhiroop Bhattacharjee, and Priyadarshini Panda. 2024. PIVOT: Input-Aware Path Selection for Energy-Efficient ViT Inference. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*. doi:10.1145/3649329.3655679.

[56] J. Duarte et al. 2018. Fast inference of deep neural networks in FPGAs for particle physics. *Journal of Instrumentation*. doi:10.1088/1748-0221/13/07/P07027.

[57] FastML Team. 2025. hls4ml. (2025). doi:10.5281/zenodo.17517206.

[58] Anouar Nechi, Lukas Groth, Saleh Mulhem, Farhad Merchant, Rainer Buchty, and Mladen Berekovic. 2023. FPGA-based Deep Learning Inference Accelerators: Where Are We Standing? *ACM Trans. Reconfigurable Technol. Syst.* doi:10.1145/3613963.

[59] Yuntao Han and Qiang Liu. 2023. HPTA: A High Performance Transformer Accelerator Based on FPGA. In *2023 33rd International Conference on Field-Programmable Logic and Applications (FPL)*. doi:10.1109/FPL60245.2023.00012.

[60] Zhen Dong, Zhewei Yao, Amir Gholami, Michael Mahoney, and Kurt Keutzer. 2019. HAWQ: Hessian AWare Quantization of Neural Networks With Mixed-Precision. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. doi:10.1109/ICCV.2019.00038.

[61] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Jill Burstein, Christy Doran, and Thamar Solorio, (Eds.) doi:10.18653/v1/N19-1423.

[62] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. 2021. Swin Transformer: Hierarchical Vision Transformer using Shifted Windows. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. doi:10.1109/ICCV48922.2021.00986.

[63] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. 2021. Training Data-Efficient Image Transformers & Distillation Through Attention. In *Proceedings of the 38th International Conference on Machine Learning*. Marina Meila and Tong Zhang, (Eds.)