Does A Fish Need a Bicycle? The Case for On-Chip NPUs in DBMS

Alexander Baumstark
TU Ilmenau, Germany
alexander.baumstark@tu-ilmenau.de

Kai-Uwe Sattler
TU Ilmenau, Germany
kus@tu-ilmenau.de

ABSTRACT

In-Database ML – integrating ML/AI features directly into DBMS – becomes more and more a requirement of customers and system builders. This includes system tasks such as indexing and query optimization, as well as application support such as inference and advanced data analytics. Although GPUs are an established technology for accelerating machine learning (ML) tasks, they are costly and require data transfer between the host and device. In contrast, neural processing units (NPUs) are becoming increasingly attractive, even as CPU on-chip extensions. In this work, we study if and how NPUs – which are dedicated to neural network operations – can be leveraged for database operations. We discuss opportunities and present results of our experiments for selected database use cases. Our findings indicate that NPUs are well-suited for latency-sensitive, small-scale ML tasks in database systems, such as learned indexes or ML-based operators.

KEYWORDS

In-Database Machine Learning, NPU, Learned Indexes

1 INTRODUCTION

Recent advances in data-driven applications have increased the demand for tighter coupling between analytics, storage, computation, and machine learning (ML) within DBMSs [17, 25, 43]. Traditional approaches, which often rely on exporting data to external ML frameworks, introduce significant latency, overhead, and operational complexity. To address these limitations, research and industry efforts have increasingly focused on integrating ML capabilities directly into DBMSs - as *In-Database ML*. This integration is now widely adopted and expected by DBMS users [9, 21], and is reshaping the architecture and functionality of modern data processing platforms [33, 38, 43, 47, 62]. This trend is supported by the emergence of ML extensions for SQL, enabling hybrid OLAP and ML workloads within a DBMS [17, 23, 31, 33].

In-Database ML. Based on these developments, the following requirements for ML–DBMS coherence can be summarized [55]: *Seamless ML integration*: Users want to train, apply, and manage models within the DBMS storage and query model [30, 33]. *Inference at scale*: There is demand for low-latency inference on fresh, real-time data, particularly in IoT or embedded DBMS [8, 30]. *Automatic and workload-driven optimization*: There is an increasing demand for auto-configuration (learned compaction [59]), and ML-based optimization for workloads [20, 57]. *Minimization of data movement*: Avoidance of costly and slow data transfers, as these are the main drivers for costs [33]. Currently, ML tasks in DBMSs, such as model inference and data transformation at moderate scales, are still mainly accelerated by GPUs. In contrast, very large-scale ML workloads are increasingly offloaded to specialized hardware stacks such as Cerebras WSEs, SambaNova RDUs, and Groq LPUs [16, 22, 48]. Although GPUs provide significant computational advantages, their integration into DBMS-based ML processing introduces several challenges: (1) *General-purpose design*: GPUs originated from graphics and evolved into broad parallel processors, with added units (e.g., tensor and ray tracing cores) increasing capability but also complexity. (2) *Data transfers*: Moving data between CPU memory, storage, and the GPU over PCIe or the network introduces significant overhead, especially for small inputs. (3) *Lower efficiency*: Their general-purpose nature leads to lower efficiency and less specialization for certain ML operations, resulting in suboptimal utilization. (4) *High costs*: GPUs remain expensive to acquire and operate, driven by high market demand and substantial energy consumption. Following the demand for ML processing capabilities, hardware vendors attempt to tackle these challenges by offering ML-specialized accelerators, referred to as *Neural Processing Units (NPU)* or *Tensor Processing Units (TPU)*, similar to PCIe add-in cards like GPUs. NPUs are designed to handle ML operator patterns (e.g., matrix multiplications, activation functions) with high efficiency, low latency and power consumption.

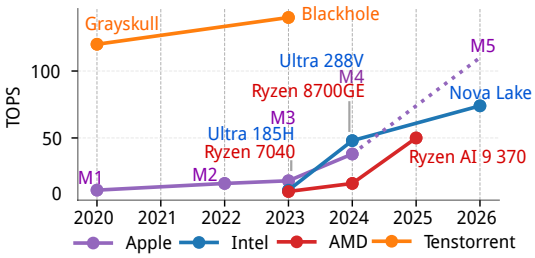

Figure 1: Achieved TOPS of on-chip NPUs on INT8 over time.

A missed opportunity. Although add-in NPUs (aNPU) have been available for several years, their adoption in DBMSs remains limited. This is primarily due to the strong software support for GPUs in ML frameworks and their generally higher performance. Primarily driven by LLM trends, a diverse landscape of NPU architectures has emerged across various vendors [1–7, 53]. A key distinguishing feature of many NPUs is the integrated *Static RAM (SRAM)*. The SRAM in NPUs serves as a cache for temporary (local) data and weight storage during processing, providing significantly higher bandwidth than even the advanced caches found in high-end GPUs. For example, the NVIDIA H100 delivers up to 6 TB/s of combined bandwidth, while the SRAM on the Tenstorrent Blackhole P150a NPU can achieve up to 46 TB/s when utilizing all 150 cores of the device [54].

 16th Annual Conference on Innovative Data Systems Research (CIDR '26). January 18-21, Chaminade, USA

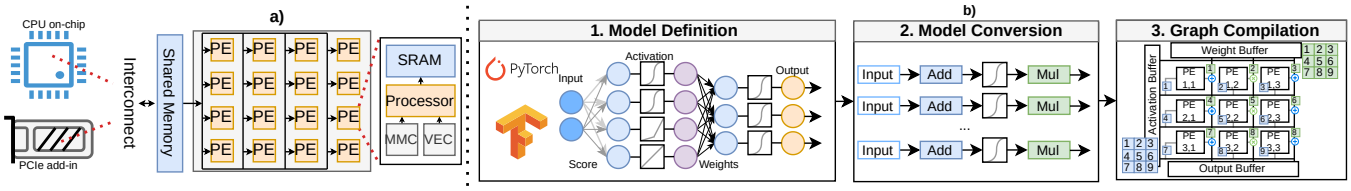

Figure 2: a) Tiled NPU architecture with PE-shared DRAM and PE-local SRAM. b) Workflow for tiled NPU architectures.

Despite this, NPUs offer far higher efficiency at much lower cost: an NVIDIA A100 delivers 624 INT8 TOPS at 600 W, while a Tenstorrent Blackhole P150a aNPU provides 664 TOPS at only 300 Watts and costs roughly 50× less.

On-chip NPUs. Recent CPU designs increasingly integrate on-chip accelerators like Intel DSA and IAA to boost memory-related tasks such as data transfer, analytics, and compression [10, 12]. This is often achieved through disintegration that enables cutting costs, increasing yields, and performance [14]. A related trend, particularly in mobile and embedded systems, is the integration of NPUs directly into the CPU SoC (oNPU), as seen in Apple M, Ryzen AI, and Intel Ultra CPUs [13, 52, 60]. These NPUs address the growing demand for real-time machine learning in consumer workloads, including LLM-based text generation, computer vision tasks like face recognition, and deep learning for predictive analytics. Figure 1 illustrates the performance improvement of already available on-chip NPUs over time. Trends in embedded CPUs, such as the AMD Versal, show an even higher increase of TOPS, reaching up to 184 [34]. As capabilities increase over time, more and more use cases for DBMS will become feasible.

Our vision: Building upon this trend, we argue that integrating an NPU into server CPUs, similar to existing DSA and IAA accelerators, could benefit ML-DBMS coherence. This argument is further supported by growing CPU-die sizes and increasingly tight CPU-ML coupling, as seen with AI-focused extensions like Intel's AMX, or disintegrated chip design [14, 44]. From a DBMS perspective, such integration could enable more efficient in-DBMS machine learning by reducing overhead for tasks like learned indexes, query optimization, and real-time inference. These models are typically small but prone to access and inference latency.

Contributions: As in analogy with the title, the prevailing view is that NPUs are as mismatched for DBMS workloads as a bicycle is for a fish [24, 28]. In contrast to this, we focus our research around the question: *What use cases and performance benefits could a DBMS gain by offloading certain tasks to an on-chip NPU?*

Therefore, our contributions and structure are as follows:
(1) We identify the challenges and opportunities enabled by current NPU architectures for DBMS in Section 2.
(2) We present use cases for DBMS where an oNPU acceleration could offer benefits in Section 3.
(3) We present initial results with real add-in cards and on-chip NPUs using realistic workloads in Section 4.
(4) We discuss related work in Section 5 and the path of realization for oNPUs and future research directions in Section 6.

2 OPPORTUNITY: NPU

We selected four representative NPU devices to explore current opportunities and challenges. For on-chip evaluation (oNPU), we chose the Intel Ultra 9 295K (oNPU(I)), and the AMD Ryzen AI 9 HX 370 (oNPU(A)). To illustrate higher compute potential, we included a *Tenstorrent* add-in card NPU (aNPU) [53].

2.1 Hardware & Workflow

Figure 2a depicts a generalized NPU architecture, derived from recent architectures [13, 34, 52, 60]. At its core, the architecture consists of a tiled array of *processing elements (PEs)*, each typically equipped with local memory or caches. Individual PEs usually contain one or more processing units, and a recent trend, driven by energy efficiency and cost considerations, is the integration of (multiple) RISC-V cores within a single PE. Local memory remains a critical component of NPU accelerators, as it has the greatest impact on overall performance. NPU accelerators primarily use small SRAM memory blocks. SRAM offers several advantages over GDDR or HBM, including the absence of constant refreshing, resulting in lower latency (1 ns vs. 10 ns), consistent timing, and improved energy efficiency. The tile-based organization of PEs further enhances data locality, making it well-suited for ML workloads, such as inference. The main drawback of SRAM is its limited capacity, so NPUs rely on PE-accessible shared memory tiles, typically backed by LPDDR. These tiles are linked through a fully configurable interconnect optimized for ML dataflows. NPUs also incorporate multiple DMA engines to enable efficient data movement.

Intel NPU. The Intel NPU (model 3720) is an on-chip NPU integrated into consumer CPUs starting with Meteor Lake, such as the Intel Core Ultra 9 285K. It delivers up to 13 TOPS at INT8 precision and is designed for power-efficient on-device inference tasks. The NPU is exposed to the host as a PCIe device and communicates through a memory-mapped, register-based interface. A LeonRT microcontroller manages runtime coordination, power states, and task scheduling on behalf of the host CPU. Internally, it comprises two Neural Compute Engine (NCE) tiles, each containing two 2000-element MAC (matrix multiply-accumulate) arrays optimized for low-precision compute, and a SPARC-based control processor responsible for instruction sequencing and synchronization. Each NCE features 2 MB of local SRAM and two dedicated DMA engines to stream data between local SRAM and shared system memory over the SoC's scalable fabric interconnect. The theoretical bandwidth to memory is 64 GB/s. On newer Lunar Lake CPUs, the integrated Intel NPU 4000 provides 48 TOPS across six NCEs and offers up to

136 GB/s of interconnect bandwidth. Upcoming Nova Lake processors are expected to include an upgraded NPU delivering up to 74 TOPS.

AMD NPU. The integrated NPU of the AMD Ryzen AI 9 HX 370 CPU is based on AMD's XDNA-2 architecture. Internally, the processing elements are VLIW cores (Xilinx Versal AIE-MLv2) running at 1.5 GHz with dedicated vector units and 64 KB of local L1 memory, arranged in a 2D compute grid. Additional memory tiles contribute 512 KB of SRAM each (L2). The detailed grid configuration is not publicly available, but it consists of 32 tiles. Based on the size of the XDNA-2 tile array in Ryzen AI devices, we estimate the total amount of on-chip SRAM in the range of 10–12 MB. So-called shim tiles provide access to shared main memory (L3) and interface the NPU fabric with the SoC interconnect. The fully programmable execution and data movement model allows explicit design of DMA transfers, routing, and inter-tile communication. In contrast to the Intel NPU, the AMD NPU allows direct configuration of the tiles to run compiled kernels in parallel. The AMD NPU delivers up to 50 TOPS on INT8 precision and can be increased by additionally leveraging the internal GPU, reaching up to 80 TOPS.

Tenstorrent NPU. For `aNPU`, we choose the devices from Tenstorrent, due to availability and more flexible software support. However, we expect similar results from devices of other vendors they share similar architectural features. The Tenstorrent architecture features multiple types of processing elements (PEs), with the primary compute PEs referred to as Tensix cores. Additional specialized cores include Ethernet cores, introduced in newer NPU generations for interconnecting multiple NPUs over Ethernet, unlike NVIDIA's proprietary NVLink, and management cores for scheduling and hardware monitoring. Each Tensix core integrates five RISC-V CPUs, vector and matrix units, data packers, and up to 1.5 MB of local SRAM.

Unlike GPU cores, which directly execute workloads, the RISC-V CPUs in Tensix cores primarily handle instruction dispatch and control flow, managing commands to the MAC, vector, and packer units. For instance, the Grayskull e150 contains 120 Tensix cores, amounting to 600 RISC-V CPUs per chip, achieving up to 332 TOPS on INT8/FP8.

Workflow & Programming model. NPUs follow the Single Program Multiple Data (SPMD) model, but with a distinct workflow. Figure 2b illustrates the workflow for NPUs.

First, a neural network model is defined or trained in a standard ML framework such as PyTorch, TensorFlow, or OpenVINO. Second, the model is converted into a compatible format (e.g., ONNX or PH), where framework-specific operations are lowered to an NPU-supported chain of primitives (matrix multiplication, convolution, etc.), dynamic shapes are replaced with static dimensions, and weights are transformed into supported data types. Finally, these primitives are compiled into NPU kernels: operators are tiled to match the PE geometry, weights and activations are assigned to device memory, and the compiler schedules data movement and on-chip dataflow across the compute cores. This enables efficient execution of complex models, as each PE can perform a distinct task, and data movement can be explicitly controlled.

This workflow is similar for most NPUs, but there are often architecture-specifics, such as distinct kernels for dataflow and computation, as for Tenstorrent NPUs.

Intel NPUs can be accessed via the Intel NPU Acceleration Library or OpenVINO, which compiles ML models into device-specific kernels by mapping operations to supported NPU operators. It supports popular frameworks such as PyTorch and TensorFlow, enabling model offloading and heterogeneous execution, e.g., so that certain layers are executed exclusively on the NPU, while the CPU handles input processing.

AMD NPUs can also be accessed through OpenVINO. For finer control over data movement, developers can use the Xilinx IRON library [27]. IRON exposes the NPU's memory hierarchy, specifically the local PE scratchpads and the dataflow between processing elements, allowing explicit management of these resources and enabling the implementation of more complex or non-standard operations. Built on MLIR and LLVM, the IRON toolchain allows the same high-level kernel code to target a broad range of AMD NPUs. Here, we see opportunities for integrating such toolchains in DBMS query pipelines via LLVM [11]. IRON provides both Python and C++ APIs and integrates seamlessly with PyTorch and TensorFlow.

The Tenstorrent architecture requires compiling three separate kernels for offloading: a reading kernel to move data from shared memory to PE memory, a compute kernel for executing workloads, and a writing kernel to transfer results back. This design ensures compatibility across current and future architectures. Tenstorrent supports MPMD execution by using separate memory banks, enabling concurrent program and model execution. Developers can use `TT-Metalium` for C++ kernels, or `TT-NN` for Python execution, supporting both TensorFlow and PyTorch model offloading.

GPU Comparison. Serving the requirements of modern workloads, GPU vendors equip their devices with *tensor cores* to provide high-bandwidth ML training and inference, in addition to the usual (Streaming multiprocessor, SM) cores. These tensor cores allow GPUs to process dense linear algebra operations, such as matrix multiplication, at higher throughput than on usual cores.

The SM cores can be utilized whenever operation complexity exceeds the capability of the tensor cores. This is mostly achieved by sharing large registers and high-bandwidth memory (HBM) between SM cores and tensor cores, leading to throughput-optimized processing. On the other hand, NPUs, especially `oNPUs`, are optimized for latency. Their grid of PEs, each paired with low-latency SRAM, enables highly predictable and efficient execution of neural network-based operations, such as convolutions, attention, or activation functions based on matrix multiplication. Due to the hardware and limitations of operator complexity, NPUs achieve a much higher energy efficiency, although lower throughput.

2.2 Opportunities

Considering the different hardware architectures of the `oNPUs` and `aNPUs`, we present in the following the opportunities of these devices based on our experiments. We further compare them with a state-of-the-art GPU (NVIDIA A100) and a CPU (Intel Xeon 9462) equipped with AMX extensions. AMX employs small 2D tile registers and an in-core fixed-function matrix-multiply unit, offering efficient acceleration but at a smaller computational scale than NPUs.

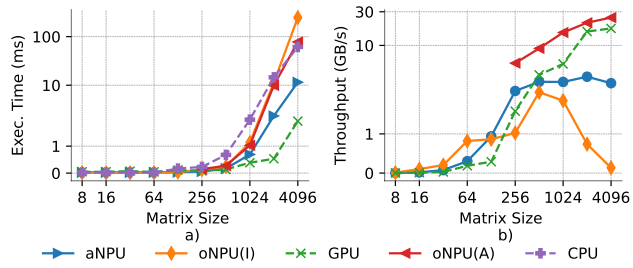

Figure 3: (a) Matrix multiplication times; (b) Host ↔ device throughput

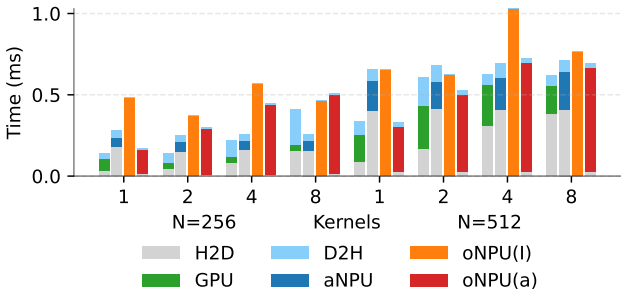

Figure 4: Execution times for running multiple kernels simultaneously.

Computation. Matrix multiplication is the main computation that can be accelerated on NPUs, which is also the basis for ML training and inference. In neural networks, fully connected layers, convolutions, attention mechanisms, and even many normalization or projection layers can be transformed into batches of matrix–matrix multiplications. Figure 3(a) presents the execution times for FP16 matrix multiplication on $B \times M \times N$ matrices across different devices.

The CPU, equipped with the AMX instruction set on Sapphire Rapids, outperforms the other devices for small matrices because it can operate directly on data resident in the L1/L2 caches, avoiding expensive transfers and benefiting from low-latency memory access. For these small workloads, the cost of kernel launch, device synchronization, and host–accelerator transfers dominates on the GPU and NPUs, while the CPU can sustain high throughput through its wide vector units and efficient cache hierarchy.

As matrix sizes increase, the oNPU(I) surpasses the CPU, as its high internal bandwidth and specialized matrix engines begin to offset transfer overheads. However, its performance drops at size 512, where the working set exceeds the internal SRAM capacity. At this point, the device must stream operands from external memory, introducing stalls and diminishing effective compute utilization.

The aNPU achieves the highest performance on large matrices due to its substantially higher compute density and memory bandwidth. It exhibits a sharp decline beyond size 4096, where memory traffic becomes the bottleneck, and GPU architectures with larger caches and more efficient scheduling become more advantageous. With the higher computation capability of the oNPU(A), a similar performance as the aNPUis possible, but slows down at higher matrix sizes due to limited internal memory.

Opportunity 1: oNPUs provide advantages for low-scale matrix computation (ML inference tasks), making them well-suited for lightweight models such as learned indexes or small language models. Here, oNPUs achieve comparable performance as GPUs.

Data transfer. Besides the CPU, all devices require a kernel and data to be transferred into their device memory. This adds overhead, especially for very low data volume. Next, we measure the combined transfer times for kernel and data to show the transfer overhead when swapping ML models. Figure 3(b) compares kernel and model transfer throughput across the evaluated devices.

The Tenstorrent device exhibits the highest transfer time across all matrix sizes, primarily due to the need to transfer three separate kernels. In contrast, kernel and data transfers to the GPU are more efficient, benefiting from a high-bandwidth PCIe interface and optimized driver for data transfers.

The oNPU(I) demonstrates the high throughput for small models, leveraging its fast CPU interconnect to achieve leading performance for sizes up to 256 (approximately 32 MB). However, as model size increases, transfer times grow substantially. This behavior is attributed to the NPU's limited on-chip memory and restricted effective bandwidth, which we measured at up to 10 GB/s. The high bandwidth interconnect of the oNPU(A) allows transferring data at a high throughput. The CPU does not require explicit data transfers due to direct main memory access. Therefore, the CPU is always more beneficial for very small and non-complex workloads.

Opportunity 2: Exploiting device-specific transfer characteristics to improve scheduling. Small models can be accelerated on oNPU due to low transfer overhead, while larger models should be routed to GPUs or high-bandwidth NPUs to maximize throughput.

Multi model. Modern workloads execute multiple ML models within a single pipeline, i.e., within a single query. Keeping several models resident on the device reduces transfer overhead but requires efficient MPMD support. Figure 4 compares different devices under MPMD workloads, where multiple independent matrix multiplications of varying sizes are executed concurrently.

For small matrices, the oNPU(I) and oNPU(A) consistently outperform the GPU, benefiting from fast memory transfers enabled by the interconnect. This can be beneficial for workloads that require low latency, like real-time ML. We focus on larger matrices (128 and 256 for one side) to better assess scalability.

As the number of concurrent kernels increases, oNPU(I) and oNPU(A) scale more efficiently, while the GPU incurs greater data transfer overhead. For example, at a matrix size of 512, the oNPU(I) begins to show a clear advantage starting from two simultaneous kernels. Since the computation capability is much lower than on GPU or aNPU, the overall execution time is much higher on oNPU, because of the data transfer. Enabled by the fast interconnect, the oNPU(A) is able to provide similar performance as the GPU when considering the data transfers. We expect increased TOPS for future oNPU devices, which further improves the performance.

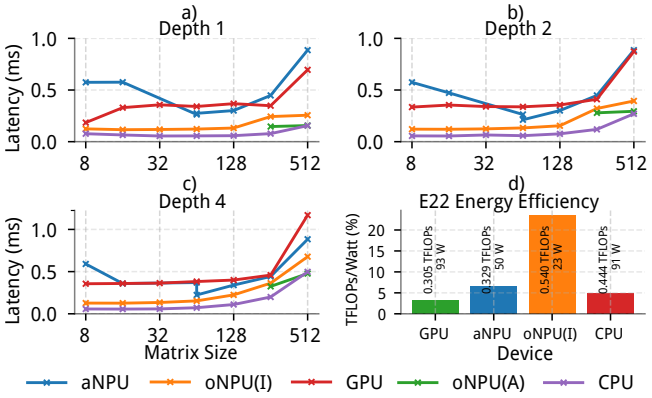

Figure 5: Inference latency on different models and inputs.

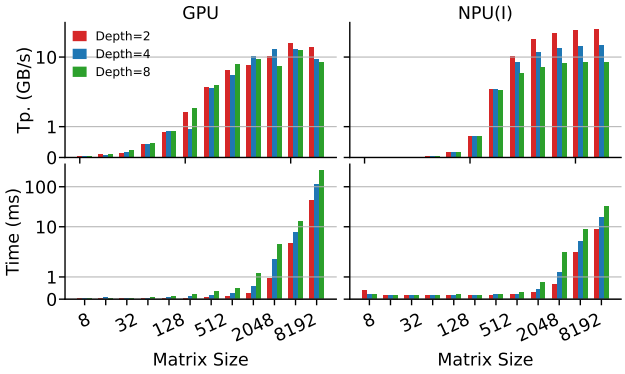

Figure 6: iGPU and NPU transfer throughput and execution time comparison of inference.

For MPMD, the aNPU behaves similarly to the GPU and achieves comparable execution times, which are constrained by the limited computation capability compared to the GPU. Future NPU designs that integrate larger on-chip memory or fine-grained activation or weight-reuse mechanisms could outperform GPUs, especially on models with extreme memory-bandwidth pressure where GPU global memory becomes the bottleneck.

Opportunity 3: Efficient support for running multiple programs or models concurrently enables high utilization of oNPU resources. For example, when serving multiple learned indexes while avoiding the overhead of repeatedly exchanging programs or model parameters between host and device.

Inference Latency. We next evaluate the inference latency of the devices, defined as the time required for a trained ML model to produce an output for a given input. To evaluate the effect of model size, we benchmark inference across different model depths (number of layers) and input sizes using a chain of matrix multiplications. For both the GPU and aNPU, data must be explicitly transferred to the device, and we include this transfer cost in the overall latencies. This reflects typical behavior during query processing, where a model may be loaded or swapped in dynamically. Figure 5(a–c) presents the latency results for the different devices and model depths. Lowest latency is achieved by the CPU due to the absence of data transfers. But across all depths, the oNPU(I) and oNPU(A) experience much lower inference latency, due to their fast interconnection and low-latency inference enabled by local SRAM. As model complexity and size increase, the runtimes on the oNPUs, CPU, and aNPU approach each other, since larger layers amortize fixed overheads and increase the share of pure computation. For the GPU, host–device data transfer dominates the runtime for smaller models, resulting in higher latencies. However, for sufficiently large models (greater than 512), the GPU's high compute throughput outweighs its transfer costs, and it delivers the lowest overall latency among the evaluated devices.

Opportunity 4: oNPU devices enable low-latency inference for small and medium-sized models, making them well suited for real-time ML workloads in DBMS settings. For very large models, GPUs remain advantageous due to their higher peak compute throughput.

Efficiency. Energy consumption is an often-overlooked factor when deploying hardware accelerators. Higher processing capability can come at the cost of increased power usage. In general, NPUs, especially oNPUs, are designed to deliver higher energy efficiency than GPUs.

In Figure 5(d), we measure the power draw (in Watts) of a single end-to-end inference on a 512×512 FP16 matrix using all available compute resources of each device and calculate the achieved energy efficiency considering the achieved TFLOPS. For the oNPUs, we report only the Intel NPU, since the AMD drivers are not providing power information for the NPUs, yet. For the GPU and aNPU, this includes input and model transfer, followed by the inference phase. Power measurements are obtained from device-level tooling such as `nvidia-smi`, `tt-smi`, and `turbostat`.

The results show that energy efficiency, measured as the ratio of achieved Watts/TFLOPS to the device's maximum Watts/TFLOPS, is significantly higher for oNPU(I) inference compared to add-in accelerators and the CPU. This advantage is largely due to the oNPU(I)'s energy-efficient components, particularly its use of SRAM. Unlike DRAM (or HBM), SRAM does not require periodic refresh operations to preserve its contents, resulting in substantially lower energy consumption. While the CPU achieves comparable inference performance, its higher power draw yields lower overall efficiency. For aNPU and GPU, the additional data transfers both reduce effective performance and increase power consumption, leading to further decreases in energy efficiency.

Opportunity 5: oNPU devices offer significantly higher energy efficiency for executing small ML models, making them well-suited for energy-constrained or high-throughput inference workloads.

Integrated GPUs. Current oNPUs share similarities with integrated GPUs (iGPUs) and are often used alongside them in mobile systems to increase processing capability further. In contrast to oNPUs, iGPUs typically do not include their own memory.

Instead, they share the same interconnect and system memory as the CPU cores. Although iGPUs are not available in server-class CPUs, we include them in our comparison with NPUs for completeness. In Figure 6, we compare oNPU and iGPU data access and processing performance using matrix multiplication.

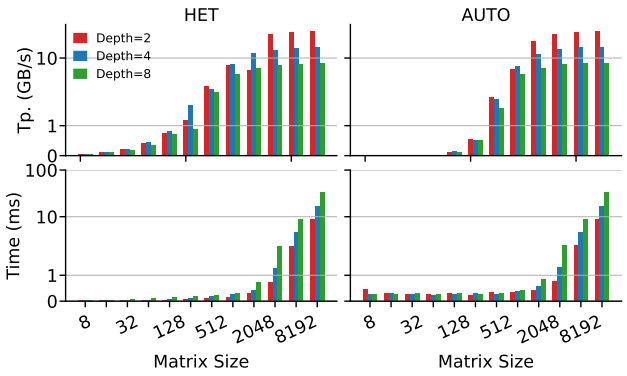

Figure 7: Heterogeneous NPU-iGPU execution compared against OpenVINO auto mode.

We focus on the Intel oNPU because the AMD devices require different frameworks to operate, while all Intel devices can be executed via OpenVINO. The iGPU achieves higher throughput for accessing model or matrix data, even at smaller model sizes, because it can directly access main memory shared with the CPU. In contrast, the NPU reaches its peak throughput only at larger model sizes, as data must first be explicitly transferred from main memory into its local memory. For compute, the iGPU is significantly faster at very small model sizes. Combined with its direct main-memory access, its performance resembles CPU-based execution but with greater parallelism. Only for larger models (≥ 512) does the NPU begin to outperform the iGPU. On NPU and smaller model sizes, the model complexity (depth) does not influence the execution time, since the PEs are not fully utilized. This gap illustrates that NPUs require sufficiently large and batched workloads to amortize data-movement overhead and expose their peak compute throughput.

Opportunity 6: NPUs deliver their best performance when data transfers are amortized.This makes NPUs particularly advantageous for workloads with sufficiently large or batched models, and iGPUs could improve computation further.

Heterogeneous Execution. In Figure 7 we compare heterogeneous NPU–iGPU execution (HET) with OpenVINO's automatic device selection (AUTO). AUTO selects the device expected to deliver the best performance at runtime and falls back if needed, whereas HET explicitly partitions the graph across multiple devices. Both configurations are offered by OpenVINO. The HET approach achieves higher throughput and lower latency by combining the strengths of both the NPU and iGPU: the iGPU excels on small inputs, while larger inputs and more complex models shift execution to the NPU, yielding superior overall performance.

Opportunity 7: Hybrid NPU–iGPU execution enables performance gains by mapping different model regions to the most suitable accelerator. This selective partitioning allows HET to surpass single backend execution, especially when workload sizes vary significantly.

Table 1: Sizes for learned indexes, optimizers, and LLMs.

Dataset	Data Size (MB)	Model Size
Learned Indexes		
IMDB - Ratings	10	30 KB
IMDB - Title	40	42 KB
TPC-H W8 - LINEITEM	240	320 KB
Learned Optimizers		
IMDB (Dace)	10	41.08 MB
IMDB (QF)	40	81.32 MB
LLMs		
tiny-random-GPT2 (rGPT2)	–	40 MB
tiny-gpt2 (tGPT2)	–	50 MB
gpt-neo-125M (nGPT2)	–	238.18 MB
facebook/opt-125m (opt)	–	477.77 MB

These opportunities reveal the benefits for oNPUs for DBMSs, especially for smaller-scale processing.

2.3 Challenges

While oNPUs offer lower throughput than GPUs, future designs are expected to support efficient processing of larger datasets. Nonetheless, our experiments reveal further challenges.

C1 Limited device memory: The limited memory of oNPUs constrains the acceleration of large workloads. While effective for small to medium workloads, their performance drops sharply once memory limits are exceeded, restricting their applicability to smaller models.
C2 Computation capability: Most NPUs are limited to matrix multiplication, which suffices for many ML tasks. However, complex workloads require transformation into matrix operations, often impractical and with uncertain performance gains.
C3 Resource utilization: As shown in our benchmarks, the overhead for processing very small data often leads to lower performance than on CPU, because the advantage of high aggregated bandwidth cannot be applied. Therefore, for very small tasks, the CPU is more suitable than the oNPU.
C4 Parallelism granularity: Due to limited abstractions, parallelism must be managed manually via workload partitioning. This increases complexity and demands explicit optimization of data layout and scheduling to prevent under-utilization.

Building on the hardware insights and identified opportunities, we now examine which database workloads can benefit from acceleration and how.

3 THE CASE FOR ON-CHIP NPUS

Given current trends in integrated accelerators (e.g., Intel DSA, IAA, AMX), tighter interconnect coupling, and rising demand for machine learning, we expect future server CPUs to integrate NPUs directly on-die, similar to current consumer, mobile, and embedded CPUs. From a DBMS perspective, this would enable more flexible and cost-effective management of ML models for diverse workloads. Figure 8a-d gives an overview of use cases in DBMSs we focus on, where we expect NPUs to improve performance or quality.

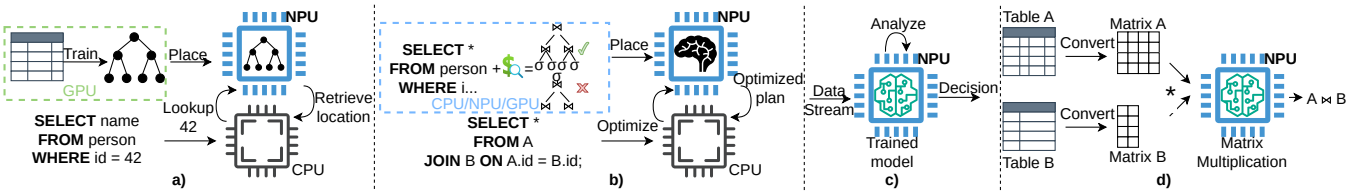

Figure 8: Identified NPU use cases in DBMS: (a) learned indexes, (b) query optimization, (c) ML operator or real-time ML acceleration, (d) offloading DBMS operators via matrix multiplication.

3.1 Taxonomy

In-Database ML approaches can be broadly categorized into two groups: *system or query optimization* and *ML operators*.

ML-based optimizers use DBMS state or user input to generate efficient execution plans, access strategies, or system configurations. These models are typically small (a few KBs to MBs), trained on internal statistics or workload patterns, and well-suited for oNPU acceleration, where low latency is critical. Similarly, ML-based cardinality estimation models are generally small and latency-sensitive, making them a natural fit for execution on on-chip NPUs. Learned indices are another strong candidate for NPU acceleration.

The ability to load models quickly from memory and support multiple models simultaneously aligns well with an oNPU and CPU interplay. On the other hand, ML-based operators such as those used for image classifications or LLM transformations tend to be larger and more complex, and are therefore better suited for execution on GPUs or high-capacity NPUs. However, recent advances in compact yet capable models (also known as small language models), such as TinyLLaMa, open up new possibilities. These models are small enough to fit on oNPU and can support lightweight text processing tasks such as filtering, generation, or information retrieval. Because these tasks are often latency-critical, fast inference capabilities are especially valuable with oNPU. Next, we discuss these use cases.

3.2 Use case: Query optimization

Learned Indexes. Learned indexes use a trained ML model to predict the location of records in memory or storage. For example, in the case of sorted data, a function can be learned using linear regression to predict the location within a small search window. A simplified conceptual overview is shown in Figure 8a.

We argue that models for learned indexes are small, even for large datasets (cf. Table 1). To evaluate oNPUs on these models, we followed the methodology of Kraska et al. [59], creating three models based on the IMDB datasets (Ratings: 10 MB, Title: 40 MB) and TPC-H (8 warehouses, 240 MB Lineitem). We trained with various configurations to optimize model size, yielding final models of 30, 42, and 340 KB. Even for large datasets, the model sizes for the learned index are small compared to a B-tree. Therefore, learned indices are perfect candidates for oNPU acceleration. Inference represents the primary advantage of using learned indices on oNPUs. Storing the learned index entirely on the NPU enables lookups to be executed with very low latency. This improves the practicality of learned approaches in DBMSs and is particularly beneficial for transactional workloads.

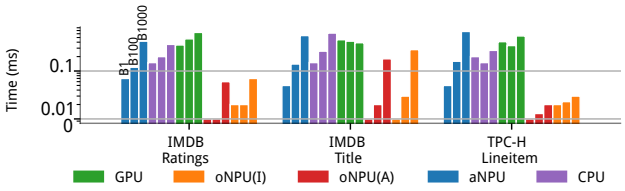

Figure 9: Learned index inference latency comparison on different devices and batch sizes (B).

Query Optimization. The effectiveness of query optimization heavily depends on accurate cardinality estimation, as it is often infeasible to compute or manage the exact statistics of (intermediate) result sets produced during query execution.

ML models for optimization are well-known and trained on queries and stats either at runtime or from prior data. A prominent example of learned query optimization is Bao [41], which not only leverages previously known results but also adapts and learns from its mistakes. Such optimizers typically require training on large datasets, so oNPUs are unlikely to offer benefits during training. We consider for our system two existing approaches for query optimization and offloading to an on-chip NPU (cf. Figure 8b).

The first is referred to as *query former (QF)* [61], which introduces a tree-structured Transformer model to represent query execution plans by capturing both operator dependencies and statistical context. This approach enables efficient cardinality estimation, cost prediction, and plan enumeration. The second query optimization model is known as *DACE* [39], and is based on the previous. It employs a lightweight tree-based Transformer to encode query plans, using tree-aware loss functions to enhance robustness. The resulting models are compact and execute efficiently on oNPUs. Trained on the IMDB with representative queries, the models are 20 and 81 MB in size.

3.3 Use case: Operators

LLMs as Operator. LLMs have gained popularity for their strong performance across diverse NLP tasks. Integrating LLMs as operators in database query pipelines enables processing of unstructured or large texts with reasoning and interpretation capabilities beyond traditional operators.

For example: (1) semantic filtering to identify the tone (angry, sarcastic), (2) summarization to reduce the text amount, (3) similarity joins using embeddings, (4) extract to filter or efficient LIKE

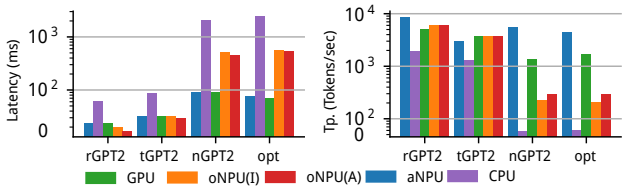

Figure 10: LLMs in query pipelines: latency and throughput.

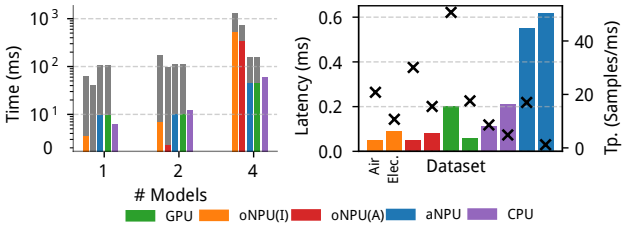

Figure 11: a) Inference on multiple loaded models. b) Real-time latency during query processing. Crosses indicate the achieved throughput.

processing. LLM inference on CPUs is limited by dedicated matrix or tensor units.

While GPUs offer strong acceleration, they incur overhead from data transfers, especially during cold starts. Their efficiency also depends on large batch sizes, making them less suited for low-latency, small-scale tasks like short text inference. oNPUs can address these challenges with low-latency cache and memory access. This enables them to execute small, frequent inference tasks efficiently with lower transfer and access overhead. As such, they are a natural fit for deploying LLMs as operators within query pipelines. A remaining limitation, however, is the constrained device memory available on current on-chip NPUs.

Many larger LLMs require hundreds of MBs to several GBs of memory. This makes careful model selection essential when targeting LLM offloading to NPUs. We select a set of current LLMs and integrate them into our prototype approach: tiny (40 MB `tGPT2`), random (50 MB `rGPT2`), neo (240 MB `nGPT2`) GPT2, and opt-125 (450 MB `opt`).

Real-time ML. Real-time ML enables inference or learning tasks to be performed immediately as new data arrives, allowing fast decision-making. This is critical for applications such as fraud detection, anomaly monitoring, and real-time recommendations. In these scenarios, inference latency determines whether the system can truly operate in real-time. Integrating real-time ML directly into a DBMS reduces data movement and simplifies system architecture by enabling models to operate close to the data.

However, GPUs present challenges in this setting because transferring data to device memory introduces additional latency. This is especially problematic for short and frequent inference tasks, such as those found in IoT or streaming environments. We see this use case as ideal for oNPUs within a DBMS. Their close integration with the CPU and access to main memory allow for small-scale inference. This makes them well-suited for embedding real-time ML operators within the DBMS, enabling more responsive execution.

DB Operators on NPU. As shown by [26], certain operators can be accelerated via tensor units of GPUs. The same can be applied to NPUs. Traditional natural joins can be reformulated as matrix multiplications by representing relations as binary indicator matrices. A conceptual overview is given in Figure 8d. Given two relations A and B with join attribute `id`, each is transformed into a matrix with rows for tuples and columns for distinct `id` values. The join result corresponds to the product $\text{mat}(A) \times \text{mat}(B)^\top$, where non-zero entries indicate join matches. Chaining matrix multiplications enables multi-way joins effectively. Aggregations can be processed via matrix reductions. This is also an appropriate use case for oNPUs. As an additional resource, it can be used to accelerate such operations, provided the data is in the same format.

The high-bandwidth interconnect of oNPUs to main memory allows for significantly improving the performance of memory-intensive operators. Their energy efficiency also makes it attractive to offload frequent, small analytical tasks at lower power cost. Many modern DBMSs already organize data in matrix-friendly layouts or can transform it efficiently, as in the case of graph CSR representations. In such settings, NPUs can accelerate join processing, aggregation operators, or graph-based traversals, reducing both latency and CPU load.

4 EVALUATION

We use the previously mentioned devices: the Tenstorrent Grayskull e150 as an add-in NPU (aNPU), the Ultra Series 295K CPU, its on-chip NPU (oNPU(I)), the AMD NPU (oNPU(A)), and an NVIDIA A100 GPU. We implement the approaches prototypically using OpenVINO for oNPU(I) and CPU, AMD IRON for oNPU(A), Pytorch via CUDA for GPU, and TT-Metalium and TT-NN for aNPU.

4.1 Learned Index & Optimization

In Figure 9, we compare the inference latency of the devices on learned index models on the dataset. We evaluated model inference across varying batch sizes to assess throughput and included model transfer times. Across all configurations, the oNPUs consistently demonstrated the lowest inference latency, ranging between 0.01 ms and 0.2 ms. For small batch sizes, the CPU exhibited lower latency characteristics compared to both the GPU and the aNPU, due to minimal invocation and transfer overhead.

As batch sizes increased, the GPU and aNPU devices achieved higher throughput, benefiting from greater parallelism and better amortization of data movement costs. These findings support our assumption: the oNPUs, with its tight coupling to the CPU and low-latency access to shared memory, is particularly well suited for small-scale, real-time inference tasks. We conducted evaluations across multiple hardware targets using the DACE and QF models. For DACE, the oNPUs achieved the lowest inference latency at approximately 0.1 ms, outperforming other devices by a factor of 3 (compared to 0.3–0.4 ms on CPU, GPU, and aNPU).

However, for the larger QF model, limited on-chip memory became a significant constraint for the oNPUs and aNPU, preventing efficient execution. In this case, offloading inference to the GPU proved more favorable due to its memory capacity.

4.2 LLMs as Operator

In Figure 10, we evaluate LLM-based operators within an end-to-end query execution pipeline. The benchmark involves a similarity filtering task over the IMDB dataset, where pre-selected keywords are matched against movie titles using embedding-based inference. We report inference latency and throughput across devices. For smaller models, the oNPUs consistently achieves the lowest inference latency, typically below 0.1 ms, benefiting from tight CPU integration and minimal data movement overhead.

Notably, even with models up to 40 MB in size, the oNPU(A) remains the fastest among all tested devices. However, as model size increases, latency grows significantly. This is primarily due to limited on-chip memory capacity and reduced compute throughput. In contrast, the aNPU exhibits improved scalability with larger models and demonstrates competitiveness with high-end GPUs when executing inference workloads involving models beyond 100 MB.

4.3 Multi-Model Processing

In Figure 11(a), we assess the multi-model inference performance across devices by concurrently executing 1, 2, and 4 models drawn from a mix of lightweight and heavyweight object detection models.

Each model is executed via a custom UDF, with parallelism achieved through independent threads. We measure the total end-to-end latency per configuration, including both model and input transfer times. Additionally, we report model transfer latency measured separately to highlight the overhead associated with cold-start scenarios where models must be loaded into device memory. For 1-2 models, the oNPU(A) delivers the lowest inference latency due to its fast access to CPU memory and minimal dispatch overhead. However, as the number of concurrently active models increases, latency on the oNPUs grow substantially, primarily constrained by limited on-chip memory capacity. In such scenarios, the GPU and aNPU exhibit better scalability and performance, even when accounting for explicit model transfer costs incurred before execution.

4.4 Real-time ML

In Figure 11(b), we measured inference latency and throughput across devices using two time series datasets: Air Quality UCI (`Air`, 1.5 MB) and Electricity Loads (`Elec.`, 250 MB). Before inference, the data were standardized using a sliding window approach with a fixed sequence length of 12. The model architecture consists of a linear projection followed by a GELU activation, representative of lightweight real-time forecasting tasks requiring low-latency inference on streaming inputs. For `Air`, the oNPUs achieve the lowest latency across all devices, outperforming even the CPU due to minimal memory transfer overhead and direct access to shared system memory. On the aNPU, the model incompatibility with the runtime API necessitated architectural adaptations, which degraded performance and resulted in the highest observed latency. For `Elec.`, the GPU demonstrates superior latency and achieves the highest throughput, benefiting from parallelism and memory bandwidth optimized for large-batch processing.

4.5 DB Operators

We implemented basic filtering, join, and aggregation operators following the approach described in [26], adapted for both the NPU and GPU backends. For evaluation, we use the IMDB *Title* and *Rating* datasets and consider the following three queries Q1 - Q3:

- **Q1:** Filter the *Rating* table by a specific rating-count predicate.
- **Q2:** Join *Title* and *Rating* to obtain the rating associated with each title.
- **Q3:** Compute the average rating across all entries in the *Rating* table.

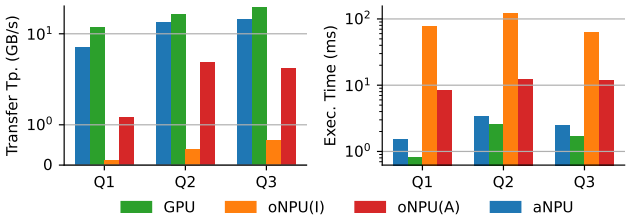

Figure 12: Transferring DBMS relations in the form of a matrix (left) and executing queries Q1-Q3 on different devices.

The results of transferring relations in the form of matrices to the devices and the execution of queries Q1-Q3 are shown in Figure 12.

Across all three database operators, we observe substantial differences in both transfer throughput and execution time between the evaluated devices. The GPU achieves the highest effective transfer bandwidth, reaching up to 12–13 GB/s. The aNPU achieves similar results but slower. The oNPU(I) achieves moderate throughput but suffers from significantly higher end-to-end latency, mostly due to lower computation capability. The oNPU(A) provides higher bandwidth than oNPU(I), especially for larger transfers (Q2, Q3), yet its throughput remains well below that of the GPU and aNPU. Since memory is limited on the oNPUs, data has to be transferred and processed in chunks, leading to much lower performance than GPU and aNPU. However, that future oNPUs can overcome this problem with faster interconnect, higher device memory capacity, and higher computation capability, making them more suited for such workloads.

4.6 Further Use Cases

Beyond these, oNPUs open up additional opportunities for accelerating database operations in further scenarios.

Streaming. High throughput and low latency are critical for streaming workloads, where data arrives continuously and must be processed in real-time. On-chip NPUs are well-suited for this scenario due to their close integration with the CPU and minimal data movement overhead, enabling low-latency inference directly on incoming data streams. Because they avoid round-trip transfers to discrete accelerators, oNPUs can sustain real-time processing for small or irregular batch sizes, which is not directly feasible with GPUs. This makes oNPUs particularly attractive for tasks such as anomaly detection, sensor analytics, where responsiveness is more important than peak throughput.

IoT. DBMS deployments in IoT environments typically operate under stringent resource and power constraints, often within

embedded systems. oNPUs provide an effective balance between computational performance and energy efficiency, enabling ML inference to be executed directly within the database engine. This reduces reliance on external servers for analysis, minimizes data transfer costs, and supports low-latency, local decision-making. By keeping computation close to the data source, oNPUs also improve privacy and robustness, since sensitive or time-critical information no longer needs to leave the device for processing.

Client-side. Current oNPUs are deployed in mobile environments such as phones and laptops. Users can leverage faster ML inference than what is typically achievable on CPUs or iGPUs, without relying on remote services. In the DBMS context, this enables interactive client-side applications such as query assistance, predictive optimization, and efficient problem diagnosis. The latter can reduce the load on customer support. Because computation remains local, NPUs also reduce bandwidth usage and mitigate privacy concerns associated with sensitive data in the cloud.

5 RELATED WORK

On-Chip Accelerators. Leveraging different on-chip accelerators for various use cases has previously been researched for DBMS. Liu et al. researched leveraging on-chip GPUs, namely Nvidia Jetson, and their use cases for DBMS [40]. Their results show that the fast interconnect and efficient scheduling allow more efficient query processing than on a CPU or a discrete GPU, due to the fast interconnect, low latency, and higher energy efficiency. Other work researched special on-chip accelerators for offloading memory movements (Intel DSA) [12] or basic filtering operations (Intel IAA) [10]. The results show that on-chip accelerators are suitable to increase throughput in memory movement, filtering, and processing compressed data. Based on this, we argue that on-chip NPUs achieve the same for ML workloads.

In-DBMS ML. Several works have already studied ML in DBMS, ranging from enabling inference directly on DBMS-stored data via UDFs, custom-made ML operators, to efficient model storage [33, 38, 47, 50, 62]. Further research proposed collocated runtimes of ML and DBMS [42, 43, 56]. Alternative approaches suggest using relational operators to perform in-DBMS inference [33, 47]. Besides ML operators and runtimes, there is research for ML-based indexing [36, 51], query optimization [19, 23], or ML-based self-tuning for DBMS [32, 49]. The majority of these works, however, consider acceleration via GPUs, which is due to the broad GPU software support of ML runtimes.

AI Accelerators. NPU devices have been available for some time, and new devices are proposed continuously, either as add-in PCIe cards [53], CPU on-chip [13, 46, 60], or embedded [34]. Datacenter NPUs are released by Google as TPUs [15]. The economic and performance advantage of the TPUs has been studied by [15, 58]. These works suggest that NPUs offer reliable performance at a lower cost for ML workloads. On-chip NPUs have been studied for the embedded and IoT use case [29]. The work suggests that on-chip NPUs have advantages, but performance limitations must be considered.

In the area of DBMS, NPU research is limited, as performance and software support are limited compared to GPUs. However, we envision that these will be changed in the future.

Tensor Processing. Still, the basis of NPUs, matrix or tensor computation, has already been proposed for DBMS using GPUs [35]. He et al. proposed transforming SQL queries into tensor programs [25] for the Hummingbird framework [43]. This is enabled by leveraging hardware-agnostic frameworks such as MLIR [37], Apache TVM [18], or OpenXLA [45]. We envision that such frameworks are perfect for acceleration with NPUs. Query operators accelerated by tensor cores of GPUs have been studied by [26]. This work shows that a high speedup is achievable when leveraging the tensor processing capabilities.

6 CONCLUSION

We demonstrated that ML inference with smaller models is particularly efficient on oNPUs. This is achieved due to the fast interconnect, providing direct access to the main memory and much lower latency than over PCIe.

However, the limited processing capability limits performance when high throughput is required, such as for model training or large batch processing. Based on this and our evaluation, we have the following lessons learned to share: (1) *Ultra-low latency* is achievable for small models on on-chip NPUs, enabling efficient cold-start inference and tight integration of ML as operators. (2) *Small input sizes* are processed more efficiently on the CPU due to lower invocation overhead. (3) *High-performance inference* requires models to be quantized in INT8 or FP16 format to match NPU-native precision, as emulated formats introduce substantial overhead. (4) *Device memory limitations* constrain both the size and number of concurrently loaded models, making resource-aware scheduling essential. (5) *Data locality* is critical, because poor memory access patterns can significantly increase latency. (6) *Custom model support* via vendor-specific software stacks can be restrictive, requiring architectural adaptations.

We argue that the following hardware capabilities are essential for making future on-chip NPUs practical and effective for DBMS integration. (1) *Higher interconnect bandwidth to main memory:* To fully exploit on-chip acceleration, the NPU must sustain high-throughput data access, minimizing bottlenecks for processing. (2) *Increased NPU-local memory capacity:* Many ML-enhanced operators require models larger than the current on-chip SRAM allows. Larger device memory enables resident models and reduces reloading overhead. (3) *Improved computational throughput:* For operator integration at scale, oNPUs must support higher FLOPs/s, parallel model execution, and better utilization for variable batch sizes to match DBMS concurrency levels. To answer the initial question, our findings demonstrate that oNPUs are well-suited for accelerating specific operations, offering performance and efficiency benefits.

ACKNOWLEDGMENTS

This work was partially supported by the DFG via the "Processing-In-Memory Primitives for Data Management" (PIMPMe) project (SA 782/31) within the "Disruptive Memory Technologies" priority program (SPP 2377).

REFERENCES

[1] 2025. Axelera AI Metis AIPU Family. https://axelera.ai/. Accessed: 2025-11-17.
[2] 2025. d-Matrix AI Acceleration Platform. https://www.d-matrix.ai/. Accessed: 2025-11-17.
[3] 2025. DEEPX NPU-Based AI Semiconductor Series. https://deepx.ai/. Accessed: 2025-11-17.
[4] 2025. Huawei Ascend NPU Series. https://www.huawei.com/en/solutions/ascend. Accessed: 2025-11-17.
[5] 2025. HyperAccel Latency Processing Unit (LPU). https://hyperaccel.ai/. Accessed: 2025-11-17.
[6] 2025. Meta Training and Inference Accelerator (MTIA). https://ai.meta.com/research/publications/mtia/. Accessed: 2025-11-17.
[7] 2025. Microsoft Azure Maia AI Accelerator. https://azure.microsoft.com/en-us/blog/azure-maia-for-the-era-of-ai-from-silicon-to-software-to-systems/. Accessed: 2025-11-17.
[8] Ashvin Agrawal et al. 2019. Cloudy with high chance of DBMS: A 10-year prediction for Enterprise-Grade ML. arXiv:1909.00084 [cs.DB]
[9] (BARC. 2024. Data Management Survey 25: Trends in Data Management, AI and Cloud Platforms. https://barc.com/news/survey-trends-data-management-2025/
[10] Alexander Baumstark et al. 2025. Uncore your Queries: Towards CPU-less Query Processing *(DaMoN '25)*. ACM, New York, NY, USA. doi:10.1145/3736227.3736243
[11] Alexander Baumstark, Muhammad Attahir Jibril, and Kai-Uwe Sattler. 2023. Adaptive query compilation in graph databases. *Distributed and Parallel Databases* 41, 3 (2023), 359–386.
[12] André Berthold et al. 2024. Demystifying Intel Data Streaming Accelerator for In-Memory Data Processing *(DIMES '24)*. ACM, New York, NY, USA, 9–16. doi:10.1145/3698783.3699383
[13] Nadav Bonen et al. 2025. Lunar Lake an Intel mobile processor: SoC Architecture Overview (2024). *IEEE Micro* (2025).
[14] Isidor R. Brkić et al. 2023. Disintegrating Manycores: Which Applications Lose and Why? *(NoCArc '23)*. ACM, New York, NY, USA, 3–8. doi:10.1145/3610396.3618090
[15] Diego Sanmartín Carrión et al. 2023. Exploration of TPUs for AI Applications. arXiv:2309.08918 [cs.AR] https://arxiv.org/abs/2309.08918
[16] Cerebras Systems. 2021. WSE. https://www.cerebras.net
[17] Cheng Chen et al. 2025. GaussDB-AISQL: a composable cloud-native SQL system with AI capabilities. *Frontiers of Computer Science* 19, 9 (2025), 199608.
[18] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Haichen Shen, Eddie Q Yan, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. 2018. TVM: end-to-end optimization stack for deep learning. *arXiv preprint arXiv:1802.04799* 11, 2018 (2018), 20.
[19] Xu Chen et al. 2023. LEON: A New Framework for ML-Aided Query Optimization. *Proc. VLDB Endow.* 16, 9 (May 2023), 2261–2273. doi:10.14778/3598581.3598597
[20] Haowen Dong et al. 2024. Cloud-native databases: A survey. *IEEE Transactions on Knowledge and Data Engineering* 36, 12 (2024), 7772–7791.
[21] Google Cloud. 2024. 2024 Gartner Magic Quadrant for Cloud Database Management Systems. https://cloud.google.com/blog/products/databases/2024-gartner-magic-quadrant-for-cloud-database-management-systems
[22] Groq Inc. 2021. The Groq Tensor Streaming Processor (TSP). https://groq.com/technology/ Accessed: 2025-07-29.
[23] Yunyan Guo et al. 2025. In-database query optimization on SQL with ML predicates. *The VLDB Journal* 34, 1 (2025), 12.
[24] Mehdi Hassanpour et al. 2021. A Survey of Near-Data Processing Architectures for Neural Networks. arXiv:2112.12630 [cs.AR] https://arxiv.org/abs/2112.12630
[25] Dong He et al. 2022. Query processing on tensor computation runtimes. *arXiv preprint arXiv:2203.01877* (2022).
[26] Yu-Ching Hu et al. 2022. TCUDB: Accelerating Database with Tensor Processors *(SIGMOD '22)*. ACM, New York, NY, USA, 1360–1374. doi:10.1145/3514221.3517869
[27] Erika Hunhoff et al. 2025. Efficiency, Expressivity, and Extensibility in a Close-to-Metal NPU Programming Interface. arXiv:2504.18430 [cs.SE] https://arxiv.org/abs/2504.18430
[28] Bongjoon Hyun et al. 2019. NeuMMU: Architectural Support for Efficient Address Translations in Neural Processing Units. arXiv:1911.06859 [cs.AR]
[29] Jovan Ivković et al. 2023. Exploring the potential of new AI-enabled MCU/SOC systems with integrated NPU/GPU accelerators for disconnected Edge computing applications: towards cognitive SNN Neuromorphic computing. In *LINK IT> EdTech International Scientific Conference. Belgrade*. 12–22.
[30] Gaurav Tarlok Kakkar et al. 2024. Hydro: Adaptive Query Processing of ML Queries. arXiv:2403.14902 [cs.DB] https://arxiv.org/abs/2403.14902
[31] Gaurav Tarlok Kakkar et al. 2025. Aero: Adaptive Query Processing of ML Queries. *Proc. ACM Manag. Data* 3, 3, Article 174 (June 2025), 27 pages. doi:10.1145/3725408
[32] Cherry Khosla et al. 2021. On a Way Together - Database and Machine Learning for Performance Tuning. In *2021 International Conference on Computing Sciences (ICCS)*. 123–128. doi:10.1109/ICCS54944.2021.00032
[33] Steffen Kläbe et al. 2023. Exploration of Approaches for In-Database ML.. In *EDBT*, Vol. 23. 311–323.
[34] Tomai Knopp et al. 2025. AMD Versal AI Edge Series Gen 2. *IEEE Micro* 45, 3 (2025), 22–30. doi:10.1109/MM.2025.3551319
[35] Dimitrios Koutsoukos et al. [n. d.]. Tensors: An abstraction for general data processing. ([n. d.]).
[36] Tim Kraska et al. 2018. The Case for Learned Index Structures. In *Proceedings of the 2018 International Conference on Management of Data* (Houston, TX, USA) *(SIGMOD '18)*. Association for Computing Machinery, New York, NY, USA, 489–504. doi:10.1145/3183713.3196909
[37] Chris Lattner, Mehdi Amini, Uday Bondhugula, Albert Cohen, Andy Davis, Jacques Pienaar, River Riddle, Tatiana Shpeisman, Nicolas Vasilache, and Oleksandr Zinenko. 2021. MLIR: Scaling compiler infrastructure for domain specific computation. In *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. IEEE, 2–14.
[38] Guoliang Li et al. 2021. AI Meets Database: AI4DB and DB4AI *(SIGMOD '21)*. ACM, New York, NY, USA, 2859–2866. doi:10.1145/3448016.3457542
[39] Zibo Liang et al. 2024. DACE: A database-agnostic cost estimator. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 4925–4937.
[40] Jiesong Liu et al. 2022. Exploring Query Processing on CPU-GPU Integrated Edge Device. *IEEE Transactions on Parallel and Distributed Systems* 33, 12 (2022), 4057–4070. doi:10.1109/TPDS.2022.3177811
[41] Ryan Marcus et al. 2021. Bao: Making Learned Query Optimization Practical. In *Proceedings of the 2021 International Conference on Management of Data* (Virtual Event, China) *(SIGMOD '21)*. Association for Computing Machinery, New York, NY, USA, 1275–1288. doi:10.1145/3448016.3452838
[42] Microsoft Corporation. 2025. Azure Machine Learning. Azure Products Page. https://azure.microsoft.com/en-us/products/machine-learning/.
[43] Supun Nakandala et al. 2020. Taming model serving complexity, performance and cost: A compilation to tensor computations approach.
[44] Nevine Nassif et al. 2022. Sapphire rapids: The next-generation intel xeon scalable processor. In *2022 IEEE ISSCC*, Vol. 65. IEEE, 44–46.
[45] OpenXLA. 2017. OpenXLA Project. https://openxla.org/xla. Accessed: 2025-01-10.
[46] Alejandro Rico et al. 2024. Amd xdna™ npu in ryzen™ ai processors. *IEEE Micro* (2024).
[47] Ricardo Salazar-Díaz et al. 2024. InferDB: In-Database Machine Learning Inference Using Indexes. *Proc. VLDB Endow.* 17, 8 (April 2024), 1830–1842. doi:10.14778/3659437.3659441
[48] SambaNova Systems. 2021. DataScale Systems. https://sambanova.ai/technology/
[49] Thomas Schmied et al. 2021. Towards a General Framework for ML-based Self-tuning Databases. In *Proceedings of the 1st Workshop on Machine Learning and Systems* (Online, United Kingdom) *(EuroMLSys '21)*. Association for Computing Machinery, New York, NY, USA, 24–30. doi:10.1145/3437984.3458830
[50] Maximilian Schüle et al. 2024. The Duck's Brain: Training and Inference of Neural Networks within Database Engines. *Datenbank-Spektrum* 24, 3 (2024), 209–221.
[51] Jiachen Shi et al. 2022. Learned Index Benefits: Machine Learning Based Index Performance Estimation. *Proc. VLDB Endow.* 15, 13 (Sept. 2022), 3950–3962. doi:10.14778/3565838.3565848
[52] Mahesh Subramon et al. 2023. AMD Ryzen™ 7040 series: Technology overview. In *2023 IEEE Hot Chips 35 Symposium (HCS)*. IEEE Computer Society, 1–27.
[53] Tenstorrent. 2025. *Grayskull™ e75/e150 Tensix Processor*. Tenstorrent. https://docs.tenstorrent.com/aibs/grayskull/ Last updated July 18, 2025.
[54] Moritz Thüning. 2024. Attention in SRAM on Tenstorrent Grayskull. arXiv:2407.13885 [cs.LG] https://arxiv.org/abs/2407.13885
[55] J Vijaya et al. 2025. Impact of Artificial Intelligence and Machine Learning Techniques in Database Management System Components. IGI GSP, 43–82.
[56] Hongyu Wang et al. 2024. Analyzing the Usability, Performance, and Cost-Efficiency of Deploying ML Models on BigQuery ML and Vertex AI in Google Cloud. In *Proceedings of the 2024 8th International Conference on Cloud and Big Data Computing (ICCBDC '24)*. Association for Computing Machinery, New York, NY, USA, 15–25. doi:10.1145/3694860.3694863
[57] Lucas Woltmann et al. 2019. Cardinality estimation with local deep learning models *(aiDM '19)*. ACM, New York, NY, USA, Article 5, 8 pages. doi:10.1145/3329859.3329875
[58] Arissa Wongpanich et al. 2025. Machine Learning Fleet Efficiency: Analyzing and Optimizing Large-Scale Google TPU Systems with ML Productivity Goodput. arXiv:2502.06982 [cs.LG] https://arxiv.org/abs/2502.06982
[59] Weiping Yu et al. 2024. CAMAL: Optimizing LSM-trees via Active Learning. *Proc. ACM Manag. Data* 2, 4, Article 202 (Sept. 2024), 26 pages. doi:10.1145/3677138
[60] Zixuan Zhang. 2021. Analysis of the Advantages of the M1 CPU and Its Impact on the Future Development of Apple. In *2021 ICBASE*. IEEE, 732–735.
[61] Yue Zhao et al. 2022. Queryformer: A tree transformer model for query plan representation. *Proceedings of the VLDB Endowment* 15, 8 (2022), 1658–1670.
[62] Zhanhao Zhao et al. 2025. NeurDB: On the Design and Implementation of an AI-powered Autonomous Database. arXiv:2408.03013 [cs.DB]