The Case for Secure Miniservers Beyond the Edge

Salonik Resch, Hüsrev Cılasun, Zamshed I. Chowdhury, Masoud Zabihi, Yang Lv, Jian-Ping Wang, *Fellow, IEEE*, Sachin S. Sapatnekar, *Fellow, IEEE*, Ismail Akturk, *Member, IEEE*, and Ulya R. Karpuzcu, *Member, IEEE*

***Abstract*—*Beyond edge devices* can function off the power grid and without batteries, making them suitable for deployment in hard-to-reach environments. As the energy budget is extremely tight, energy-hungry long-distance communication required for offloading computation or reporting results to a server becomes a significant limitation. Based on the observation that the energy required for communication decreases with shorter distances, this paper makes a case for the deployment of *secure beyond edge miniservers*. These are strategically positioned, lightweight local servers designed to support beyond edge devices without compromising the privacy of sensitive information. We demonstrate that even for relatively small scale representative computations – which are more likely to fit into the tight power budget of a beyond edge device for local processing – deploying a beyond edge miniserver can lead to higher performance. To this end, we consider representative deployment scenarios of practical importance, including but not limited to agricultural systems or building structures, where beyond edge miniservers enable highly energy-efficient real-time data processing.**

***Index Terms*—Homomorphic encryption, intermittent computing, processing in memory.**

I. Introduction

BEYOND edge devices can operate without batteries or other constant power sources and rely only on energy harvested from their environment [14]. These devices can reside in a variety of environments including building structures (to monitor structural integrity or to detect anomalous events) [24], farming fields (to monitor soil moisture, temperature, crop growth and other parameters enabling precision farming) [32], remote wilderness [14], disaster sites (to monitor and estimate the status of the disaster to coordinate relief efforts) [45], outer space (as disposable nanosatellites) [23] or the human body (as implants) [16].

While this extreme deployment capability opens up many new applications, beyond edge operation is subject to a stringent power budget. The power budget depends on the deployment environment. Thermal energy harvesters can provide a power budget of $60\,\mu\mathrm{W}/$ [18], which can reach $100\,\mathrm{mW/cm^2}$ for solar harvesters [15]. Energy efficiency dictates the performance [14], as typically most of the time is spent in waiting to harvest sufficient energy.

The stringent power budget makes it impossible to perform all necessary steps of computation beyond the edge. Accordingly, beyond edge devices can only perform very limited local computation and have to send data to a server for further processing. Not surprisingly, however, the stringent power budget also severely limits data communication. If the server is a few kilometers away, data communication cost can reach $400\,\mu\mathrm{J}/$ per bit [5]. Sending even a single sample of the MNIST data set [20] in this case can consume $2.5\,\mathrm{J}/$. The data transfer overhead can dramatically reduce with increasing proximity of beyond edge devices to the respective server. For example, within tens of meters Bluetooth Low Energy (BLE) requires as little as $158\,\mathrm{pJ}/$ per bit [36]. *Offloading computation is only feasible if the server is close enough to beyond edge devices.*

Clearly, considering environmental constraints and typical deployment scenarios, it is not feasible to place full-fledged servers within tens of meters of beyond edge devices. To be more specific, any server in close proximity to beyond edge devices would be subject to similar environmental constraints, and therefore, by itself has to conform to beyond edge operation semantics. In this paper, we will refer to such local beyond edge servers as *miniservers*.

Whileminiservers are subject to strict power budgets as well, by definition, they should be allocated a higher power budget than the beyond edge devices (such as sensors) they are servicing. For example, a miniserver can be placed on open land in the crop field with abundant sunlight, providing a milliwatt-range power budget. Using this relatively larger power budget, the miniserver can perform computation for nearby beyond edge devices which are more power-constrained. Devices with much lower power budgets (such as sensors) spread out in crop fields shaded by crops, receive less sunlight and harvest less energy. When needed, they can offload their computations to the miniserver. While the miniserver has significantly reduced performance when compared to a standard (cloud-based) full-fledged server, it is designed to survive in extreme enviroments such that it can reside in close proximity to beyond edge devices.

Being deployed beyond edge in potentially insecure environments, the miniserver can easily be compromised or stolen, making data stored on the miniserver (such as machine learning models) vulnerable. Exposing the miniserver only to encrypted data through homomorphic computing can eliminate such vulnerabilities. The challenge then becomes fitting the

Received 6 November 2024; revised 11 June 2025; accepted 24 June 2025. Date of publication 16 July 2025; date of current version 10 September 2025. Recommended for acceptance by T. Adegbija. *(Salonik Resch and Hüsrev Cılasun contributed equally to this work.) (Corresponding author: Hüsrev Cılasun.)*

Salonik Resch, Hüsrev Cılasun, Zamshed I. Chowdhury, Masoud Zabihi, Yang Lv, Jian-Ping Wang, Sachin S. Sapatnekar, and Ulya R. Karpuzcu are with the University of Minnesota, Minneapolis, MN 55455 USA (e-mail: cilas001@umn.edu).

Ismail Akturk is with Ozyegin University, Istanbul 34794, Türkiye.

Digital Object Identifier 10.1109/TC.2025.3589691

significant overhead of homomorphic computing into the extremely constrained miniserver power budget. To address this challenge, in this paper we propose to (i) leverage extreme efficiency of nonvolatile processing in memory (PiM) for beyond edge deployment; (ii) restrict the type of homomorphic computations on the miniservers to a less energy-hungry but common subset.

First, power sources beyond the edge are not reliable and operational interruptions are common. Unlike volatile memory systems, which lose stored information during a power outage, non-volatile PiM retains data without external power [35]. Therefore, checkpointing, the process of saving the system state to guarantee forward progress after interruptions, becomes significantly more efficient. Second, performing only common linear operations on the miniserver, such as multiplication and addition, can significantly reduce the homomorphic overhead. This is a good compromise as long as most of the operations in the target computation are linear – which is the case, e.g., in support vector machine (SVM) inference as a representative application. At the same time, keeping the computational depth low reduces noise, which in turn eliminates the need for *bootstrapping*, one of the most energy-intensive components of homomorphic computing.

Recognizing that communication overhead increases proportionally with distance, this study advocates for the strategic deployment of lightweight miniservers beyond the edge to securely serve adjacent beyond-edge devices. These miniservers, endowed with access to superior power resources owing to their geographic positioning, leverage highly energy-efficient, intermittent-resilient nonvolatile processing-in-memory technology to facilitate homomorphic operations that ensure data privacy. Using representative deployment scenarios including agricultural environments and building infrastructures, we demonstrate that even relatively modest computational tasks, aligned with the stringent power constraints of beyond-edge devices, exhibit improved performance when offloaded to such miniservers.

In the following, we will analyze challenges and opportunities to make the case for secure miniservers beyond the edge. We will start with the definitions, problem statement, as well as design space exploration in Section II; and continue with background information on homomorphic computing (Section III-A), SVMs (Section III-B) and nonvolatile PiM (Section III-C). We will cover the system architecture and operating semantics in Section IV; quantitative characterization, in Sections V and VI; a discussion and summary of our findings, in Sections VII and IX; and related work, in Section VIII.

II. Design Space

In the remainder of this paper we will refer to each beyond edge device to be serviced as a FLY; each beyond edge miniserver as a RAT; and each full-fledged, tethered remote server, as a TURTLE.

A. Definitions & Problem Statement

TURTLE is a distant server in a secure environment with continuous power and an order of magnitude higher power budget than FLY or RAT (there is no need for energy harvesting). It is the control center and the intended final destination for any result of interest.

FLY is a beyond edge device running on harvested energy. It acquires data with sensors, with the intention of reporting interesting results back to the TURTLE. FLY is deployed in a hard-to-reach (or concealed environment), such as crop fields (or within the concrete blocks of a bridge at the time of construction), where it can monitor parameters such as soil moisture, temperature and carbon dioxide concentration (or vibrations to trigger warnings about structural integrity of the bridge). Because it is physically concealed, FLY can operate on raw (unencrypted) data. Due to its environment, however, FLY has restrained capability to harvest energy (such as solar, RF or thermal) which severely constrains the power budget, typically to the µWatt range [18].

RAT, the miniserver, also operates beyond the edge and relies on harvested energy. Unlike FLY, however, it is deployed in a location where it can harvest more energy, such as on open land (or on top of the bridge) with direct exposure to sunlight. RAT's location allows for higher capacity energy harvesting compared to what is available to FLY, which directly translates into a higher power budget. For example, during daylight hours RAT can harvest sunlight to operate at a power budget of tens of milliwatts [18]. However, due to being exposed in the field (or on top of the bridge), RAT is not protected from potential adversaries, as opposed to TURTLE, thus, cannot be trusted. Data stored on RAT, such as propriety machine learning models or analysis results, are vulnerable. One way to preserve privacy in this case is letting RAT process data only using homomorphic computing (Section III-A).

Fig. 1 provides an overview for cases where (a) FLY is untrusted, and (b) FLY is trusted: in both cases TURTLE is d_{FT} meters away from FLY, and it takes E_{FT} µJ per sample to transmit data from FLY to TURTLE. RAT is d_{FR} meters away from FLY and it takes E_{FR} µJ per sample to transmit data from FLY to RAT. Necessarily, $d_{FT} >> d_{FR}$ as RAT has to reside near FLY. By construction, FLY can communicate more efficiently with RAT than with TURTLE.

B. Design Space Exploration

Without loss of generality, in the following we assume that FLY continuously acquires sensor data (to be processed using machine learning inference) from its environment. We are after the most energy efficient design which preserves privacy. FLY has three options:

Option-1: Send all sensor readings to the distant TURTLE. TURTLE will be in charge of all computation (inference).

Option-2: Perform all computation (inference) locally and only send meaningful results to TURTLE.

Option-3: Offload computation (inference) to the nearby RAT and then only send meaningful results to TURTLE.

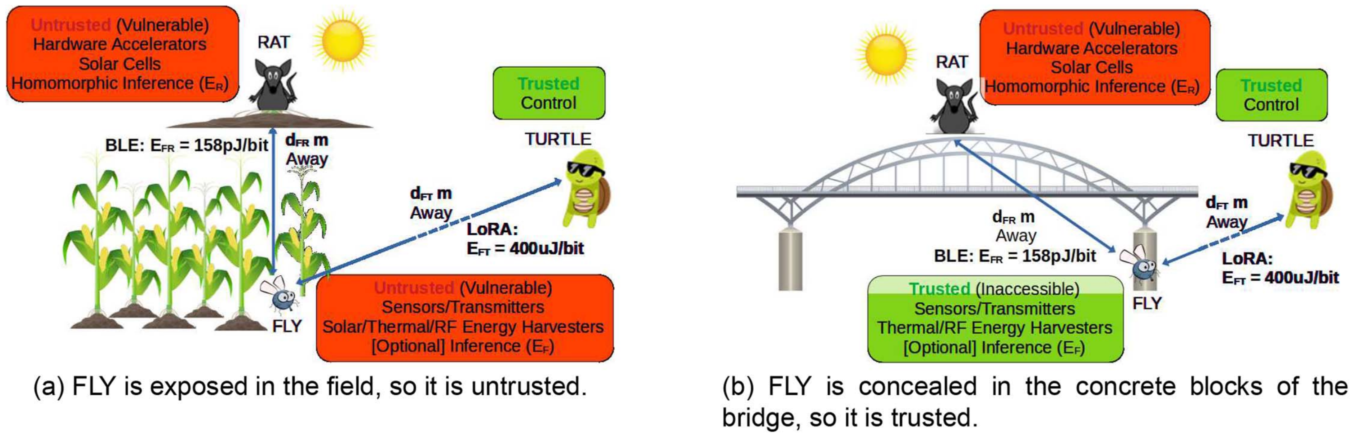
Untrusted (Vulnerable)
Hardware Accelerators
Solar Cells
Homomorphic Inference (E_R)
RAT
BLE: E_{FR} = 158pJ/bit
d_{FR} m Away
Trusted Control
TURTLE
d_{FT} m Away
LoRA: E_{FT} = 400uJ/bit
FLY
Untrusted (Vulnerable)
Sensors/Transmitters
Solar/Thermal/RF Energy Harvesters
[Optional] Inference (E_F)
(a) FLY is exposed in the field, so it is untrusted.
Trusted (Inaccessible)
Sensors/Transmitters
Thermal/RF Energy Harvesters
[Optional] Inference (E_F)
(b) FLY is concealed in the concrete blocks of the bridge, so it is trusted.

Fig. 1. FLY can offload computation to nearby RAT as needed. Final destination of results is the distant TURTLE. LoRa: Low-power long-range communication.

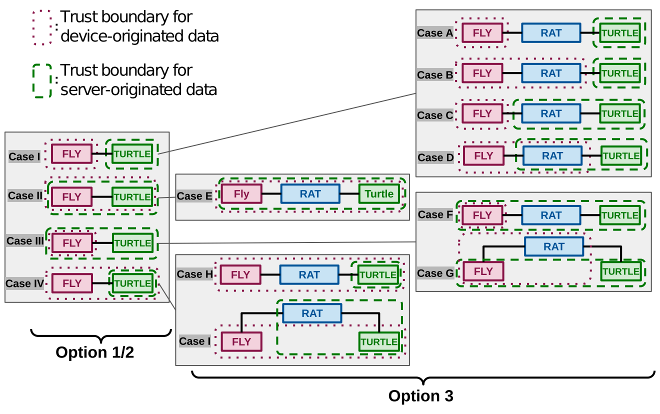

Fig. 2. Possible trust boundaries for **option-1/2** and **option-3**: an overview.

Threat Model & Trust Boundaries: *The feasibility of each of these options depends on how trust boundaries of FLY, RAT, and TURTLE are aligned.* The **trust boundary** delimits data communication between trusted parties. Communication within a trust boundary may or may not be encrypted, yet all parties within a trust boundary eventually have access to unencrypted information. On the other hand, all information must remain encrypted outside the trust boundary. We assume that the communication links are trusted.

Fig. 2 tabulates all possible alignments of trust boundaries for all three options. Any untrusted device storing data (computing) without (homomorphic) encryption exposes critical information. To preserve privacy, encryption or homomorphic computing is necessary outside the trust boundary with respect to the origin of the data. FLY-originated information includes sensor data or telemetry information collected by FLY. TURTLE-originated data includes the inference model, which may contain confidential intellectual property.

Option-1 or **Option-2** only involve a beyond-edge device (FLY) and a full-fledged server (TURTLE). In **Case-I**, there is no overlap between the trust boundaries of FLY and TURTLE, hence, any data communication between FLY and TURTLE has to be encrypted, regardless of where the computation happens. **Case-II**, on the other hand, covers the other extreme with perfectly aligned trust boundaries, which obviates the need for communication or processing of encrypted data. **Case-III** (**Case-IV**) covers an asymmetric alignment of trust boundaries, where FLY (TURTLE) cannot trust TURTLE (FLY) and accordingly data outside FLY (TURTLE) has to be encrypted.

Even for **Case-II,** *which does not incur any overhead for privacy preservation,* **Option-1** *typically is the least efficient among other options due to the higher energy cost per bit of communication compared to computation. Additionally, since the data primarily comes from the sensors and the information content of raw (unprocessed) sensor data usually is low, transmitting all acquired data without any preprocessing to TURTLE is wasteful [14].*

Let us assume that inference on FLY takes E_F µJ per sample; the probability that a data sample contains results of interest (to be sent to TURTLE) is α, and that **Case II** applies. Then, performing inference locally on FLY (**Option-2**) would be more efficient than **Option-1**, if

$$E_F + \alpha \times E_{FT} < E_{FT}. \quad (1)$$

This inequality almost always holds as relatively energy costly long distance communication beyond 3km with low-power long-range (LoRa) hardware can consume up to $400\,\mu$J per bit [5]. For example, sending data samples (such as MNIST image samples) would consume $E_{FT} = 2.5\,$J per sample. In comparison, the energy required for local inference per sample (with standard beyond edge hardware) can get as low as $E_F = 84.1\,\mu$J [13]. Even for a more conservative estimate of $E_F = 27\,$mJ [14], local inference by FLY would be more efficient if $\alpha < 98.9\%$. Provided that results of interest are relatively rare, $\alpha < 98.9\%$ is almost guaranteed to be the case.

Option-2 *is typically is more promising than* **Option-1** *for beyond edge operation. The complication with* **Option-2** *is the extremely constrained power budget of FLY which can limit the scope of local computations. Only* **Option-3** *involves a beyond-edge miniserver (RAT) which can extend the scope of local computations beyond the capabilities of FLY.*

Without loss of generality, we will use the representative deployment scenario from Fig. 1(b) as an illustrative example for **Option-3**, where FLY and TURTLE are trusted. This corresponds to **Case-B** from Fig. 2: TURTLE data – including the TURTLE-sourced machine learning model – cannot leave TURTLE unencrypted and must be processed homomorphically outside TURTLE.

The key difference between **Option-2** and **Option-3** is in where computation (inference) takes place. In most strict terms, if we keep everything for **Option-2** intact (including the data transfer between FLY and TURTLE) except the local inference on FLY, **Option-3** would span

1) Transmission of raw sensor data from FLY to RAT (consuming E_{FR} per sample)
2) Encryption of sensor data on RAT (consuming $E_{encrypt}$ per sample)
3) Homomorphic inference on RAT (consuming E_R per sample)
4) Transmission of results from RAT back to FLY (consuming E_{RF} per sample)
5) Decryption of results on FLY (consuming $E_{decrypt}$ per sample)

In this representative case for **Option-3** (Fig. 1(b)), the asset to be protected is the machine learning model used to perform inference on RAT. The main threat stems from the exposed nature of RAT, which makes the hardware vulnerable to attacks – the attacker can even gain full physical access. The mitigation mechanism involves storing the machine learning model encrypted in RAT. The homomorphic computing setting here allows RAT to encrypt the data received from FLY, which is sufficient to perform inference homomorphically. However, only FLY and TURTLE are trusted, and hence only FLY can decrypt data.

In this case, for **Option-3** to be more favorable over **Option-2**, RAT should meet the following energy (Eqn.2) and time (Eqn.3) constraints, where P_F is the power available to FLY; and P_R, to RAT:

$$E_{FR} + E_{\text{encrypt}} + E_R + E_{RF} + E_{\text{decrypt}} < E_F \tag{2}$$

$$\frac{E_{FR}}{P_F} + \frac{E_{\text{encrypt}}}{P_R} + \frac{E_R}{P_R} + \frac{E_{RF}}{P_R} + \frac{E_{\text{decrypt}}}{P_F} < \frac{E_F}{P_F} \tag{3}$$

Formally, this decision problem can be expressed as a binary optimization problem as follows:

$$\begin{aligned}&\min_{x_2,x_3\in\mathbb{B}}\\ &-x_2\left(\frac{P_F(E_{\text{tiny encrypt}}+E_R+E_{RF})+P_R(E_{FR}+E_{\text{tiny decrypt}})}{P_FP_R}\right)-x_3\frac{E_F}{P_F}\\ &+2x_2x_3\left(\frac{P_F(E_{\text{tiny encrypt}}+E_R+E_{RF})+P_R(E_{FR}+E_{\text{tiny decrypt}})}{P_FP_R}+\frac{E_F}{P_F}\right)\end{aligned} \tag{4}$$

x_2 $(x_3) \in \mathbb{B}$ here is a binary decision variable, which, when set to logic 1 indicates that **Option-2** (**Option-3**) should be selected. The cost function then becomes the overall power consumption, and the term with x_2x_3 multiplier penalizes solutions with both decision variables being true simultaneously. Although the cost function is quadratic, it has a trivial solution since $x_2, x_3 \in \mathbb{B}$, and as a result, we can optimize power by simply comparing the power consumption of both options.

E_{encrypt} and E_{decrypt} can consume as low as $0.06\,\text{mJ}/$ using specialized hardware for homomorphic encryption on edge devices [42]. The close proximity of FLY to RAT (in the order of meters), significantly reduces the communication cost. Technologies, such as Bluetooth Low Energy (BLE), become applicable with some configurations offering as little as $158\,\text{pJ}$ per bit [36]. FLY can use BLE for communication with RAT; and LoRa, for communication with the distant TURTLE. E_R, the homomorphic computing energy on RAT, is a key component in Eqn.2. While RAT has a higher power budget than FLY, the overhead of homomorphic computing can still be challenging.

Considering these challenges and opportunities, to better put viable design options into perspective, we will first cover background information. We will then propose an efficient beyond-edge miniserver architecture for RAT and quantitatively compare and contrast its feasibility to alternatives.

III. Background

A. Homomorphic Computing

RAT, designed as a beyond edge miniserver, is intended for deployment in a range of environments that may be susceptible to adversary attacks. There is also the risk of RAT being relocated or stolen. This is especially concerning as RAT may store proprietary machine learning models. Given the potential for physical access by an adversary, traditional security measures often fall short. Homomorphic computing, which ensures that sensitive data remains encrypted at all times, is particularly well-suited to address these challenges. Thus, we assume that all sensitive data (i.e., the machine learning model in our illustrative case study) on RAT is maintained in an encrypted state. Homomorphic encryption enables direct computation on encrypted data. Since the machine learning model remains encrypted throughout RAT's lifetime and RAT does not have the key for decryption, the model is protected against physical attacks on RAT.

Homomorphic computing incurs significant time and energy overhead [6], however, linear operations typically are more efficient than other computational primitives. Accordingly, to keep the energy overhead at bay, RAT only performs linear operations *homomorphically*; and FLY, the remaining few non-linear operations *non-homomorphically*. This execution model is practical, as linear operations (e.g., multiplications and additions) dominate the operations in compact applications like Support Vector Machines (SVMs) suitable for beyond edge deployment. (Section III-B).

The first step under homomorphic encryption is encoding a data vector into a plaintext, which represents a set of coefficients in a polynomial. This plaintext is then encrypted into a ciphertext. The number of coefficients, i.e., the degree of the polynomial modulus, sets the length of the plaintext and ciphertext. Each step of homomorphic computation introduces noise on the encrypted data, which accumulates over multiple steps of computation. Excessive noise can prevent correct decryption, hence noise limits the number of homomorphic operations performed on the encrypted data. The state-of-the art noise reduction technique, bootstrapping, interleaved with actual homomorphic computing steps, can help, but is a known performance bottleneck. Considering the tight power budget due to beyond edge operation, we do not consider bootstrapping or any technique to prevent noise accumulation. Generally, a higher degree of the polynomial modulus translates into a higher value for the maximum number of permissible homomorphic operations before the onset of destructive noise accumulation. We use BFV for integer arithmetic on RAT [6], which is available on Microsoft SEAL [38]. Testing with Microsoft SEAL [38], we determine the minimum vector size to successfully complete SVM computations as 4,096. The coefficient modulus is the maximum value of the ciphertext coefficients, represented in a residue number system consisting of three 36-bit prime numbers. A ciphertext consists of two such polynomials. Hence, the total memory required by the ciphertexts becomes $2 \times 3 \times 36 \times 4096 \sim bits = 108 \sim KB$.

Homomorphic addition is mainly based on element-wise additions of the vectors involved. Moreover, routine modulus operations are necessary to prevent coefficients from growing excessively large. Because coefficient modulus is known ahead of time, we can implement modulus with efficient in-memory shift, add, and subtract operations [26]. Homomorphic multiplication, on the other hand, is significantly more complex. The primary components are the number theoretic transform (NTT), mathematically equivalent to integer Fast Fourier Transform (FFT), along with scaling and base conversions [41]. We follow the computation steps laid out by Özerk et al. [29] for NTT; and by Al Badawi et al. [2], for the rest.

B. Support Vector Machines

Support Vector Machines (SVMs) are popular machine learning (ML) algorithms used for classification and regression tasks. They are particularly effective for smaller datasets and can perform well in beyond edge environments where computing resources and power budget are limited. The advantages of SVMs include a strong theoretical foundation and proven performance on a wide variety of classification problems [28]. SVMs are trained by inspecting input samples for a problem to identify samples indicative of each class. After training is complete, these samples form the *support vectors*. New inputs are classified based on their similarity to the support vectors from each class. By construction, a standard SVM model produces a binary classification. We use a simple extension to multiple classes by training a separate SVM for each class. For example, to classify MNIST, we use 10 SVM classifiers, each identifying one digit, ranging from 0 to 9.

SVM inference primarily boils down to the dot product of each input sample with each of the support vectors. The dominant operations (multiplications and additions) are linear. At the very end, the dot products are squared (multiplication by self), summed, and finally compared. This final comparison forms the only non-linear operation. While complete SVM training and inference has been successfully demonstrated using homomorphic operations [30], we opt to perform only the initial multiplications and summations on RAT. FLY performs the final squaring, summing and comparison, which, if performed homomorphically, would require highly expensive operations such as rotations to sum elements in the same ciphertext [33]. This ensures that we do not exceed the maximum number of homomorphic operations (i.e., maximum permissible computational depth) to avoid destructive noise accumulation, as we do not perform bootstrapping.

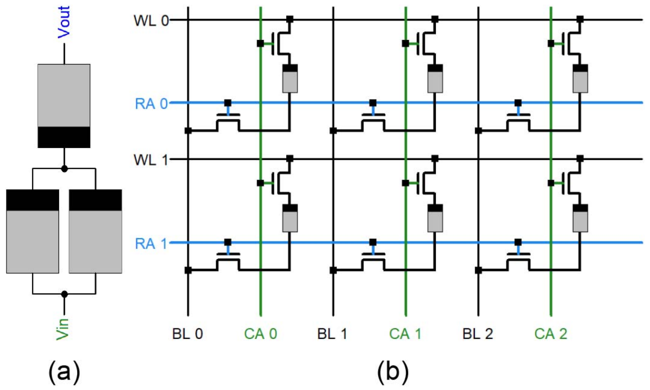

Fig. 3. (a) Boolean gate formation. Each rectangle corresponds to an MTJ, with fixed (free) layer marked in lighter (darker) shade. Input MTJs (in parallel) are connected in series with an output MTJ. (b) 2T(ransistor)-1M(TJ) PiM array. CA: Column activate; RA: Row activate; BL: Bitline select; WL: Wordline select.

C. Nonvolatile Processing in Memory (PiM)

Nonvolatile (resistive) PiM architectures, such as MOUSE [35] that strictly perform all computation within the memory array, are very well suited to beyond edge deployment due to extreme energy efficiency and intermittent-safe operation under frequent power outages. Therefore, without loss of generality, we base the hardware design of RAT on nonvolatile spintronic PiM.

Magnetic Tunnel Junctions (MTJs) represent the core building blocks for spintronic PiM. MTJs are nonvolatile (resistive) memory devices consisting of two magnetic layers, a *free layer* and a *fixed layer*. The polarity of the free layer can change, while the polarity of the fixed layer remains intact. When the magnetic layers are (anti)aligned, the MTJ is in the (anti)parallel state and has a (high) low resistance (R_{AP}) R_P. The resistance levels encode logic values 0 and 1. Passing a sufficiently large current ($> I_{switch}$) through the MTJ from the free (fixed) layer to the fixed (free) layer can bring the MTJ to the (anti)parallel state.

All it takes to perform logic gates with MTJs is [8]: (1) Connecting MTJs to serve as inputs and as the output to form a resistive divider as shown in Fig. 3(a); (2) Presetting the output to a known (gate-specific) logic value; (3) Applying a gate-specific voltage across the resistive divider from Fig. 3(a) such that the output changes state, i.e., switches, according to the truth table of the target gate. To be more specific, the cumulative induced current through the inputs depends on the resistance, hence logic state, of the inputs, and may or may not exceed I_{switch} to switch the output. For example, we can

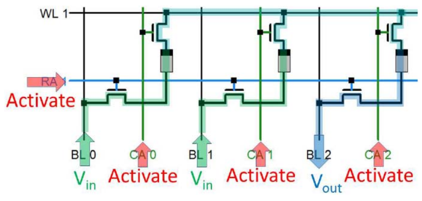

Fig. 4. V_{in} and V_{out} applied to the bitlines drive logic operations. Row activate (RA) and column activate (CA) select active rows or columns. Shown is a row-logic operation implementing the gate from Fig. 3(a).

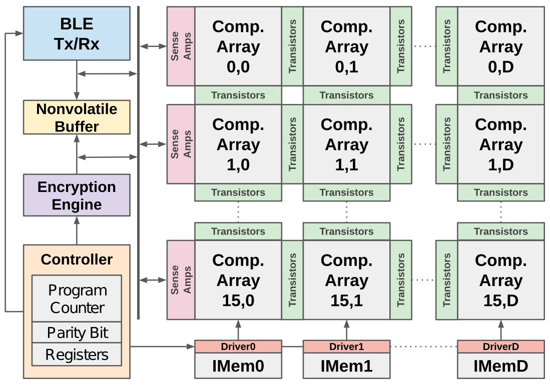

Fig. 5. RAT architecture. BLE Tx/Rx: Bluetooth low power transmitter/receiver. IMem: Instruction memory.

perform a two-input NAND by (1) Following the topology from Fig. 3(a); (2) Presetting the output to 0 (R_P); (3) Applying a NAND-specific voltage such that electrons flow from the inputs to the output. If both (either of) input MTJs are (is) in $1/R_{AP}$/high resistance ($0/R_P$/low resistance) state, the current through the output would be $< (>) I_{switch}$ and the output remains 0 (switches to 1).

Using this principle for gate formation, the array architecture of MOUSE [35] allows only one gate operation to be performed in a column at a time, but all rows can perform the same operation in parallel. While RAT uses the same principle for gate formation, the underlying architecture is more flexible, in that gates can be formed both within rows and columns, by applying voltages along the bitlines (BL) or the wordlines (WL), as depicted in Fig. 3(b) [34]. The underlying PiM architecture is a fundamental difference between MOUSE and RAT. RAT's PiM arrays essentially boil down to digital crossbars, but without the infamous sneak paths which can compromise correctness. Access transistors controlled by row activate (RA) and column activate (CA), remove potential sneak paths from the array [34]. Fig. 4 shows an example for row-logic implementing the gate from Fig. 3(a).

RAT can perform many logic operations in each row (column) simultaneously, as long as the inputs and the output reside in the same columns (rows). Having support for both row- and column-logic facilitates more efficient data movement within the array when compared to the standard, energy-hungry memory read and writes. We will refer to rows or columns performing computation as *active*. Accordingly, in row- (column-) logic, we need to specify the column (row) addresses of the inputs and the output for each operation, along with a list of active rows (columns).

IV. RAT: A Homomorphic Beyond Edge Miniserver

A. RAT Miniserver Architecture

As shown in Fig. 5, RAT features non-volatile memory arrays augmented with computation capability (computation arrays) and banks of non-volatile instruction memory with volatile circuitry for driving operations.

1) Computation Arrays: As primary components of RAT, they store the encrypted data and perform all homomorphic computation. Computation arrays are organized in a mesh configuration, where only arrays in the first column are equipped with sense amplifiers which enable direct memory read and writes. For the rest of the computation arrays, inter(intra)-array data transfers boil down to inter(intra)-array logic operations [34]. Transistors between neighboring arrays act as switches to stitch arrays together such that they can participate in the same logic operation [34].

As shown in Fig. 3(b), each cell in a computation array has one MTJ and 2 access transistors. For reference, featuring a different array architecture, MOUSE uses only one transistor per cell [35]. Two transistors significantly increase the area per cell from $0.0012\,\mu\mathrm{m}^2$ [25] to $0.038\,15\,\mu\mathrm{m}^2$ [43], but they are instrumental in removing crossbar sneak paths. More area efficient approaches to eliminate the sneak paths would not fit into the tight power budget of beyond edge devices [21]. As a result, for the smallest benchmark we consider in this paper, ADULT, the homomorphic computation arrays occupy $6.72\,\mathrm{mm}^2$. For the largest benchmark, MNIST, the area expands to $377\,\mathrm{mm}^2$. For reference, TI-MSP430FR5994 microcontroller, commonly used as a sub-component of beyond edge systems [14], occupies roughly $100\,\mathrm{mm}^2$.

We use ciphertexts of 4,096 elements (Section III-A). Hence, it is ideal to allocate 4,096 rows in order to store all elements and to perform parallel element-wise homomorphic multiplications and additions using row-logic. However, due to parasitic bitline/rowline resistance and capacitance, computation array size is limited to 1024x1024 [43]. Hence, we use 16 rows of computation arrays, where each array is 512x512. We utilize a row of transistors to conditionally connect the bitlines of neighboring computation arrays, effectively forming larger arrays [34]. Under this configuration, simple logic operations suffice to transfer bits between adjacent arrays.

The most space intensive step of homomorphic SVM computation is the initial multiplication, which may involve either two ciphertexts or a ciphertext and a plaintext. We assign a single element of each polynomial to each row. Hence, each row requires $2 \times 2 \times 3 \times 36 = 432$ bits for ciphertexts, 36 bits for *twiddle factors* of NTT operations [29], plus additional bits for temporary workspace. This easily fits within two columns of computation arrays (1024 bits). However, we still need a third column of computation arrays to store additional twiddle factors required for different stages of the NTT algorithm. In total, $log_2(4096) = 12$ twiddle factors are required in each row.

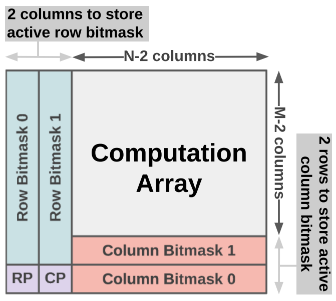

Fig. 6. RAT dedicates two rows and two columns of each computation array to bitmasks indicating active rows and columns. Standard logic and write operations can set the bitmasks. Dedicated instructions configure the peripheral circuitry according to the bitmasks. Two copies of bitmasks guarantee intermittent-safe operation along with a row parity (RP) and a column parity (CP) bit.

As RAT can use logic operations to transfer data within and between computation arrays, we only need to read the cells which store the final result (and which RAT in turn sends to FLY). There is no need for sense amplifiers for the vast majority of the computation arrays. It suffices to keep sense amplifiers only for the computation arrays residing in the first column of the mesh.

Computation arrays must also maintain the active set of rows or columns. One option is using two external non-volatile registers in each computation array – with as many bits in the registers as there are rows and columns in a computation array, to act as a bitmask for logic operations. We instead dedicate a single row and column in each computation array to keep this information. Dedicated instructions use these “registers” to activate the peripheral circuitry. To guarantee correctness under intermittent operation (Section IV-B3), RAT keeps two copies of these (Fig. 6).

2) Drivers and Instruction Memory: For every operation in each array, we need to specify the type of the logic gate to be performed (which determines the gate-specific voltage across bit/row lines and the output preset), the addresses of the input(s) and the output, and the set of rows and columns to be activated. This gives rise to an instruction set architecture similar to MOUSE[35], however, due to the difference in the array architecture, the interface is different: As each array can compute independently, we need to specify many different gates, inputs, outputs, row and column bitmasks simultaneously. To efficiently distribute the corresponding signals, we group computation arrays into columns. Computation arrays in the same column act as a single unit and are driven by the same control signals. For each column of computation arrays, there is an associated *driver* and a non-volatile *instruction memory* array.

Instruction memory arrays store the operations that each column, formed by computation arrays, is supposed to execute – such as NOT, (N)AND, (N)OR – along with the row/column indices of the input(s) and the output for every operation. Each instruction contains (i) an opcode specifying the logic operation (3 bits); (ii) up to three addresses for input(s) and output (12 bits each); (iii) whether the operation is to be performed within a row or a column (1 bit).

All drivers operate synchronously and initiate computation within the computation arrays. After receiving a command from the controller, each driver retrieves an instruction from the specified address, decodes it, and sends the input and output addresses to the row/column decoders. The driver then applies the specific gate voltage along the bit and row lines of the computation arrays.

3) Encryption Engine: RAT operates on encrypted data to protect its ML model (encrypted at all times). Per our threat model (Section II), RAT encrypts the data received from FLY before processing. Homomorphic encryption engines for edge devices exist [42], and RAT relies on such a design.

4) Receiver and Transmitter: Without loss of generality, RAT uses Bluetooth Low Power (BLE) to communicate with the nearby FLY. BLE devices can offer extremely energy efficient communication at short distances [22]. We use a configuration of particularly low power, down to 158 pJ per bit [36]. BLE supports communication within a few tens of meters.

5) Controller: A dedicated controller orchestrates control and data flow by maintaining a status register (SR) and a program counter (PC), according to the algorithm in Fig. 7 and the corresponding state transition diagram from Fig. 8. It also activates the BLE transmitter/receiver and the encryption engine, and sends trigger signals to the drivers, which in turn execute logic operations within computation arrays.

B. Operation Semantics

The algorithm in Fig. 7 and the state transition diagram in Fig. 8 summarize RAT’s operation semantics. RAT supports fine-grain checkpointing in the *Compute* state to guarantee forward progress in the presence of frequent power interruptions. Checkpointing in the *Compute* state is very efficient due to unique physical characteristics of MTJ devices [35]. Checkpointing in other states is less efficient, leading to more wasted work per restart. However, since RAT predominantly operates in *Compute* state, this overhead remains tolerable.

1) Data Reception and Transmission (I/O): Due to intermittent power, RAT cannot rely on continuous communication. The communication protocol is simple: When RAT is not busy (there is no input being processed), RAT waits for new input from FLY. After completing the reception of the input, RAT enters the *Encrypt* state and stops checking for new inputs, not even acknowledging them. Following encryption, RAT performs the computation in *Compute* state. Once this is finished, RAT reactivates the BLE to transmit outputs to FLY.

A dedicated non-volatile buffer keeps input data (Fig. 5). The size of the input data is fixed apriori, so RAT knows the number of packets to expect. A dedicated region of memory keeps each packet. There is a *valid bit* for each packet. RAT sets all valid bits to 0 prior to reception. The valid bit for each packet is set strictly after the packet gets written into memory. If RAT looses power after the packet is written but before setting the valid bit, the packet is considered invalid and needs to be re-sent. Once all valid bits are set (to indicate

Operation Semantics

```
function ReceiveInterruptHandler
    $valid[packetID] \leftarrow True$                              ▷ Set packet valid bit
end function
function TransmitInterruptHandler
    $sent[packetID] \leftarrow True$                               ▷ Set packet sent bit
end function
function LowPowerInterruptHandler
    $Restart \leftarrow True$                                      ▷ Set restart bit
end function
while $True$ do                                                    ▷ Main loop body
    $RxComplete \leftarrow False$                                  ▷ Enter RECEIVE state
    $valid[packetID] \leftarrow False\ \forall packetID$
    while $\overline{RxComplete}$ do                               ▷ Receive state
        if $valid[packetID] = True\ \forall packetID$ then
            $RxComplete \leftarrow True$                           ▷ Indicate receive complete
        end if
    end while
    $EncComplete \leftarrow False$                                 ▷ Enter ENCRYPT state
    $Restart \leftarrow False$
    while $\overline{EncComplete}$ do
        Encrypt
        if $Restart \neq True$ then                                ▷ Ensure restart at interruption
            $EncComplete \leftarrow True$
        end if
    end while
    $parity \leftarrow 0$                                          ▷ Enter COMPUTE state
    while $PC \neq End$ do
        $Restart \leftarrow False$
        ComputeInstruction
        if $Restart \neq True$ then                                ▷ Ensure restart at interruption
            $PC \leftarrow PC + 1$
            $parity \leftarrow !parity$
        end if
    end while
    $TxComplete \leftarrow False$                                  ▷ Enter TRANSMIT state
    $Restart \leftarrow False$
    $sent[packetID] \leftarrow False\ \forall packetID$
    while $\overline{TxComplete}$ do                               ▷ Transmit
        if $sent[packetID] = True\ \forall packetID$ then
            if $Restart \neq True$ then                            ▷ Ensure restart at interruption
                $TxComplete \leftarrow True$                       ▷ Indicate transmit complete
            else
                $sent[packetID] \leftarrow False\ \forall packetID$ ▷ Retransmit
            end if
        end if
    end while
end while
```

Fig. 7. RAT's operation semantics. The pseudecode closely follows Fig. 8 which consists of four states, namely RECEIVE, ENCODE, COMPUTE, and TRANSMIT. The interrupt handlers ReceiveInterruptHandler, TransmitInterruptHandler, and LowPowerInterruptHandler serve the receive/transmit interrupts per processed packet as well as the interrupts due to power loss. Packet progress is tracked using *valid*/*sent* bits that are indexed with $packetID$, $*Complete$ signals triggering switching to the next state. $Restart$ is used to reprocess an interrupted operation.

that all packets are received), RAT sets a *completed* bit, which moves RAT controller to the next state. During data reception and transmission, no other components of RAT are in use. The controller stalls all operations until data transfer is complete.

2) Encryption: To be intermittent-safe, all activity of the encryption engine has to be atomic. The encryption engine reads input from the dedicated non-volatile buffer, performs encryption, and writes the result into the computation arrays in the first column (Fig. 5). Atomicity implies that if the encryption engine is interrupted at any point, it restarts from the very beginning. This guarantees correctness on restart, as none of the inputs can be overwritten, however, it can waste energy. There is significant room for optimization when it comes to

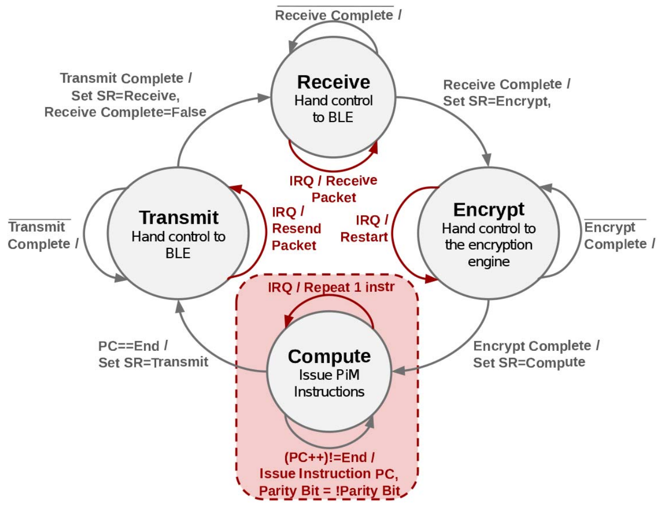
Receive Complete /
Receive
Hand control to BLE
Transmit Complete / Set SR=Receive, Receive Complete=False
Receive Complete / Set SR=Encrypt,
IRQ / Receive Packet
Transmit
Hand control to BLE
Transmit Complete /
IRQ / Resend Packet
IRQ / Restart
Encrypt
Hand control to the encryption engine
Encrypt Complete /
IRQ / Repeat 1 instr
PC==End / Set SR=Transmit
Compute
Issue PiM Instructions
Encrypt Complete / Set SR=Compute
(PC++)!=End / Issue Instruction PC, Parity Bit = !Parity Bit

Fig. 8. Operation semantics. The controller is responsible for maintaining architectural state variables and ensuring correctness of state transitions. SR: Status register. PC: Program counter.

augmenting the encryption engine with a more efficient checkpointing mechanism which is beyond the scope of this work. RAT spends the vast majority of time and energy in the homomorphic computation arrays, which feature efficient checkpointing. The encryption engine writes its output directly into computation arrays, hence data becomes ready immediately for processing once encryption is complete.

3) Computation: Computation takes the most time and energy, therefore it is critical to make it resilient to intermittent operation without compromising energy efficiency. Prior work has demonstrated that in-memory logic performed with MTJs is inherently intermittent-safe [35]. We exploit the same feature to enable frequent but very light-weight checkpointing during computation. However, unlike prior work, we do not need to maintain standard memory functionality for most of the computation arrays and can simplify the hardware accordingly.

During computation, the controller broadcasts the PC value to all drivers. Each driver reads the corresponding instruction in its instruction memory, decodes it, then drives the appropriate signals to the computation arrays in the same column. Sufficient time is allotted between PC value broadcasts such that every instruction has enough time to complete. This process repeats either until the program finishes or until RAT runs out of power.

Writes and logic operations with MTJs remain correct when interrupted or performed multiple times [35] – irrespective of when the interrupt happens, be it right before the operation starts, in the middle of processing or after the operation finishes. In other words, each single instruction represents an *idempotent* operation. Therefore, after RAT restarts, the controller can safely send the same PC value to the drivers. Due to this inherent resilience, guaranteeing correctness in computation arrays becomes straight-forward as long as we stick to the program order by performing a single instruction at a time. This translates into checkpointing after every instruction and not starting the next instruction until the previous one is complete. Checkpointing after each instruction may be inefficient for more traditional architectures, however, as RAT performs all computation in (non-volatile) memory, all data backup happens automatically. All results are permanently stored after every instruction, regardless of the checkpointing strategy. Hence, RAT can frequently checkpoint with low overhead by simply tracking architectural state variables in the controller. For computation, the controller only needs to keep track of the valid PC value.

Architectural state contains bitmasks that keep track of the active rows and columns in each computation array (Section IV-A1). As bitmasks reside in computation arrays in dedicated rows and columns, they are protected by the same mechanism. Being non-volatile, bitmasks persist through power interruptions. However, the peripheral circuitry needs to be reactivated on restart. Dedicated instructions, *activate rows* and *activate columns*, restore peripheral circuitry according to the bitmask values. These instructions are issued on restart and whenever the bitmasks change.

An advantage of storing the active row/column bitmasks in the computation arrays is that they can be set with standard write and logic operations. Such operations may get interrupted before finishing, after changing only some bits in the bitmask. When the operation is repeated on restart, potentially corrupted bitmasks hence can compromise correctness. Our solution is to duplicate the bitmasks and maintain two parity bits, one row parity (RP) and one column parity (CP), which indicate which bitmask is valid (Fig. 6).

4) Control: RAT's controller is responsible for orchestrating the state transitions from Fig. 8 and for maintaining architectural state variables required for driving computation: the *status register* (SR), which indicates whether RAT is in the *Receive*, *Encrypt*, *Compute*, or *Transmit* state; in addition to the *program counter* (PC), which is sent to the drivers to initiate instructions in the *Compute* state.

Given the relatively little information the controller must maintain, simple checkpointing strategies are sufficient. Accordingly, SR and PC are duplicated and each have an ancillary non-volatile parity bit. The parity bit indicates which copy is valid, and is flipped strictly after each update to SR or PC completes. Setting the parity bit is an atomic operation – the MTJ holding the parity bit either successfully flips or keeps its old value, if interrupted during an update. If RAT restarts without flipping the parity bit, the old values of SR and PC apply, and work gets repeated. If RAT restarts after flipping the parity bit, effectively all progress has been saved and RAT starts from where it left off. This all suffices for correctness, because everything in RAT – instructions, state transitions of the controller, or updates of the architectural state variables – is sequential and proceeds strictly *in order*.

During *Transmit* and *Receive* states, the controller hands control over to the BLE transmitter/receiver, and waits for the *Complete* signal, which triggers the transition to the next state. If interrupted before the arrival of *Complete*, RAT hands control over to the transmitter/receiver on restart. In the worst case, an entire data packet needs to be re-transmitted or received. Additionally, if transmission/reception completed, but the *Complete* signal was not received prior to shut down, the controller re-checks the signal before progressing to the next state. An efficient protocol for intermittent safe transmission/reception is beyond the scope of this work (Section VII), where general strategies addressing noisy transmission with high packet loss rates [4] are applicable. Encryption follows a similar strategy. If there is a restart in the *Encrypt* state, the controller instructs the encryption engine to start from the beginning.

RAT checkpoints after every instruction only in the *Compute* state. The controller sends the instruction address to the drivers, which load the instruction and start it in the computation arrays. The controller waits a sufficiently long time to guarantee the completion of the instruction, after which it updates the PC and commits the instruction by flipping the non-volatile parity bit. As this natural "checkpointing" happens after each instruction, at most one instruction needs to be re-performed in the case of a power outage. The same RAT instruction, when repeated multiple times, always produces the same result.

V. Evaluation Setup

In the following, we will provide a quantitative characterization for the beyond-edge miniserver RAT by using the representative deployment scenario from Fig. 1(b) (**Case I** from Fig. 2) as an illustrative example. The deployment scenario dictates the relative alignment of respective trust boundaries, and therefore, the local computation vs. communication trade-off.

Without loss of generality, we experiment with homomorphic SVM inference on RAT as a sufficiently compact computation suitable for beyond edge operation. We use MNIST [20], Human Activity Recognition (HAR) [3], and ADULT [19] datasets, representative of input sizes to fit in beyond edge devices. These datasets cover different tasks, such as image classification and activity recognition, which fit the capabilities of RAT. When performing homomorphic inference, the data size is determined by D, the dimension of the input samples, and N, the length of the homomorphic ciphertexts. D individual elements from each sample reside in separate ciphertexts as each element gets summed together (elements within the same ciphertext cannot be added without expensive rotation operations [33]). As described in Section III-A, for our ciphertexts $N = 4096$. The number of support vectors required for each benchmark fills the 4096-element ciphertexts, with the remaining elements padded with dummy data. Hence, regardless of the number of support vectors, each benchmark uses 4096-element ciphertexts. MNIST contains 784 elements (representing a 28×28 image); HAR, 561 elements; and ADULT, 14 elements. We use 3-bit integers for all benchmarks. This lowers transmission cost and prevents arithmetic overflow during computation. We customize SVM models to use only integer arithmetic and limit the number of support vectors to 4096. Despite these limitations, we are still able to achieve reasonable accuracy relative to full-precision SVMs provided by libSVM [7], which we run on R using the e1071 package. SVM parameters and accuracy results are listed in Table I. We use BFV encryption for integer arithmetic [6], which has a lower overhead. We validate our implementation on Microsoft SEAL [38].

We model the energy harvester as a constant power source, which fills a $1\,\mathrm{mF}/$ capacitor (energy buffer) on the chip. In practice, tunable energy buffering systems such as Capybara

TABLE I
SVM Benchmarks and Rat Parameters. The Number of Support Vectors Does Not Impact Latency or Energy as They Are Padded to 4096. For Comparison, We Also Report the Accuracy of Full-Precision SVMs From LibSVM [7] With Unlimited Support Vectors

	D	# Support Vectors	RAT Accuracy	libSVM Accuracy
MNIST	784	3841	93.85%	98.05%
HAR	561	2466	94.64%	94.1%
ADULT	14	596	76.00%	78.62%

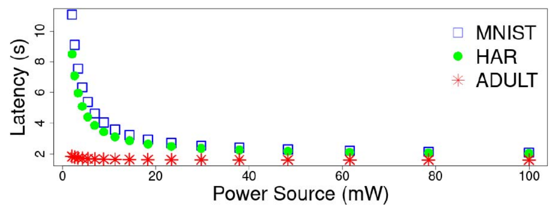

Fig. 9. Latency of homomorphic SVM inference on RAT when using different power sources. A larger capacity power source (x-axis) results in a larger power budget, hence less frequent power-downs. As a result, the latency (y-axis) to finish computations reduces.

[9] can be used. The power source raises the voltage on the capacitor to a specified level. Once this level is reached, RAT activates and begins consuming energy until the voltage on the capacitor drops to the minimum threshold. RAT then shuts down and waits for the capacitor to recharge. We assume that RAT has an ability to harvest energy from relatively higher capacity power suppliers for beyond edge devices (e.g., sunlight). We test with up to 100 mW (1 cm^2 solar panel [18]). It is desirable to match the voltage level on the capacitor to the MTJ technology used. For SOT MTJs, the voltage fluctuates between 200 mV and 450 mV. As noted in Section III-C, different logic operations require different voltages. We use switched-capacitor converters for upconversion and downconversion to provide all required voltages [17].

MTJ based memory technology is already commercially available [1], however, the devices are expected to significantly improve over the next few years. We use the MTJ model parameters from [44]. We develop a performance simulator to quantify latency and energy, which accounts for control and peripheral circuitry overheads extrapolated from NVSIM [12]. NVSIM reports the relative share of latency and energy of the peripheral circuitry for non-volatile memories of the same size as computation arrays. We conservatively clock RAT to give more than sufficient time to the controller, to the drivers, and to the arrays to finish logic operations in the memory.

As discussed in Section IV-B3, logic operations in memory are inherently resilient to interrupts and we can practically checkpoint after every instruction. This doesn't apply to data reception/transmission and encryption. For transmission/reception, we assume that an interrupt results in the retransmission or reception of a single packet of data. Packets transmitted prior are saved in non-volatile memory. For encryption, we assume that the *entire* process restarts if interrupted. At restart, the capacitor is fully charged and RAT has sufficient energy to complete the process. We assume 158 pJ per bit [36] for transmission cost. We model the latency and energy of the encryption engine after Van der Hagen et al. [42] – 0.3 ms and 60 μJ – which operates on ciphertexts of the same size as in our work.

VI. Evaluation

We will first characterize the latency of end-to-end homomorphic computing on RAT as a function of the frequency of interrupts due to power outages, which is a function of (decreases with increasing) harvested power budget. We will thereby identify the *minimum harvested power budget* necessary for homomorphic computing on RAT (**Option-3** from Section II) to become faster than pure local computation on FLY (**Option-2** from Section II). We will demonstrate that this power budget matches the power budget of practical energy harvesters to make the case for RAT. For the respective power budget, we will continue with a detailed characterization of RAT's performance and energy along with a quantitative demonstration of its efficiency for intermittent computing.

Figure 9 captures how end-to-end inference latency (including transmission and encryption overhead) on RAT evolves as a function of power source capacity (budget). In intermittent computing systems energy efficiency dictates the time to finish computations, as these systems typically spend most of their time powered-off waiting to harvest sufficient energy [14], [35]. A larger capacity power source (x-axis) results in a larger power budget, hence less frequent power-downs. As a result, the latency (y-axis) to finish computations reduces. A lower capacity power source on the x-axis incurs a lower power budget, and a higher number of interrupts due to power outages, which in turn increases the latency to finish computations. Due to the high energy cost of homomorphic computing, the latency is quite high and increases dramatically as the power source capacity reduces. We deem RAT feasible if the latency under a reasonable power budget remains less than the latency of pure local processing on FLY.

To estimate the latency for local processing, we base FLY on the beyond edge accelerator RipTide [13], which represents a highly energy efficient design from recent literature. The more energy-efficient FLY is, the less likely it is that a miniserver like RAT is needed. As FLY's power source is pretty limited by construction (e.g., to thermal energy harvested within the walls of a bridge where FLY is deployed), we can assume a reasonable power budget of roughly 60 μW [18]. Using the data and comparison baselines (which better align with our benchmarks and datasets) from the RipTide paper [13], and assuming iso-accuracy, we estimate the performance of RipTide-based FLY as tabulated under **Option-2** in Table II: If FLY is as efficient as RipTide, it would take 0.49 s to complete MNIST; 1.42 s, HAR; and 0.055 s, ADULT. Under **Option-1**, assuming a data transmission cost from FLY to TURTLE of approximately 400 μJ per bit, FLY would take 15,680 seconds for MNIST; 11,220 seconds for HAR; and 280 seconds for ADULT – clearly much longer than **Option-2**. Matching **Option-2** latencies from the

TABLE II
THE CASE FOR RAT: FLY CAN REPORT ALL DATA TO THE REMOTE SERVER TURTLE (OPTION-1); COMPUTE LOCALLY (OPTION-2), OR OFFLOAD COMPUTATION TO THE NEARBY BEYOND EDGE MINISERVER RAT (OPTION-3). IF FLY IS AS ENERGY-EFFICIENT AS RIPTIDE [13] WITH A REASONABLE 60 µW POWER BUDGET (WELL WITHIN REACH OF THERMAL ENERGY HARVESTERS [18]), OPTION-1 AND OPTION-2 RESULT IN THE REPORTED LATENCIES IN THE FIRST TWO COLUMNS. WE ASSUME ENERGY PER BIT FOR TRANSMISSION TO THE DISTANT SERVER TURTLE TO BE 400 µJ [5]. ALSO SHOWN IS THE POWER BUDGET REQUIRED BY RAT TO MAINTAIN FASTEST OPERATION UNDER OPTION-3

	Option-1 TURTLE	**Option-2** FLY	**Option-3** RAT
	Latency E_{FT}/P_F (s)	Latency E_F/P_F (s)	Min. power required for fastest operation (mW)
MNIST	15,680	0.49	$\approx$ 100
HAR	11,220	1.42	
ADULT	280	0.055	

TABLE III
ENERGY CONSUMPTION OF POLYNOMIAL MULTIPLICATION. N: POLYNOMIAL SIZE; B: BITWIDTH

Architecture (N,b)	Energy (μJ)
X86 (1K,16) [26]	2483.77
X86 (4K,32) [26]	10864.64
FPGA (1K,16) [27]	12.52
CryptoPIM (1K,16) [26]	11.04
CryptoPIM (4K,32) [26]	178.62
RAT (1K,16)	2.42
RAT (4K,32)	13.66

second column of Table II with RAT's latencies from Fig. 9, we can derive the necessary power budget for RAT to render **Option-3** faster than **Option-2**: ≈ 100mW across the board. By construction, in comparison to FLY, more power is available to RAT, which can typically reach tens of milliWatts [18]. 100mW is within the capacity of modern solar panels [15]. *In a nutshell: Using typical energy harvesters,* **Option-3** *can result in faster operation than* **Option-2** *even for an extremely energy efficient FLY design (based on RipTide in our case study) and for a conservatively clocked RAT (Section V).*

We will next take a closer look into RAT's performance and energy consumption. Consistent with prior work [14], [35], energy efficiency determines latency as RAT is energy constrained. Efficient checkpointing makes RAT's energy consumption more sensitive to the length of the program, rather than the number of power interrupts. To evaluate the efficiency of homomorphic computing on RAT, Table III provides a comparison to representative CPU, FPGA [27], and PiM (CryptoPIM) [26] implementations for polynomial multiplication – the core of homomorphic multiplication and the most energy intensive arithmetic component of our benchmarks. RAT provides a significant advantage over CPU, and consumes less energy than FPGA and CryptoPIM for the same problem size. Even though RAT represents the most energy-efficient solution, we should note that this comparison for even the same problem size is not fair, as none of these alternatives is intermittent-safe and can reliably support beyond edge deployment under **Option-3**. Guaranteeing resilience under intermittent operation can have a sizable performance and energy overhead, which we will characterize next. For reference, the absolute energy consumption of end-to-end SVM inference on RAT takes 1 188 716.63 µJ for MNIST; 851 282.72 µJ for HAR; and 31 736.27 µJ for ADULT, respectively.

To evaluate the impact of intermittent operation, we next examine the *dead*, *restore*, and *backup* overheads [37] in terms of latency and energy (Fig. 10). *Dead* refers to the overhead for re-performing operations after restart which were already

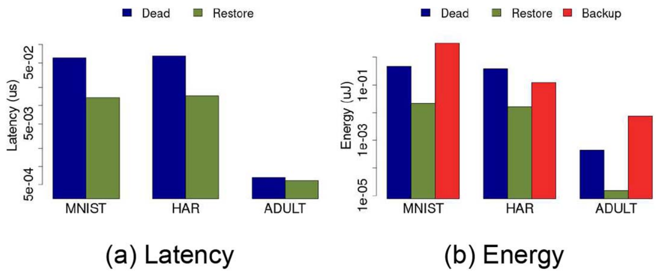

Fig. 10. RAT: latency and energy overhead of intermittent-safe operation.

performed before shutdown. This accounts for wasted operations – which completed but the results of which cannot be used. For RAT, this translates into re-performing the last in-memory instruction before shut-down, re-transmitting or receiving a packet of data, or re-encrypting input data. *Restore* refers to the overhead of restarting the device, getting it back into working order after a power outage. For RAT, this is the re-activation of rows and columns of the computation arrays. *Backup* covers any operation performed in order to save state prior to shutdown. This typically involves saving the architectural state and storing volatile data to non-volatile memory. As RAT performs all computation in memory, data backup occurs automatically. Hence, the only backup cost for RAT is saving the architectural state, and setting the SR, PC, and parity bits. We report these overheads at the lowest power considered (Table II), which results in the maximum number of restarts, and therefore, the maximum overhead.

Fig. 10(a) characterizes the energy; Fig. 10(b), the latency overheads during RAT's intermittent operation. Overall, we observe that all overheads remain practically negligible, showcasing the efficiency of RAT as a beyond edge miniserver. To be more specific, dead energy is 0.0040% of the total; restore energy, 0.0002%; and backup energy, 0.0147%. Dead energy reflects the overhead of re-performing instructions (typically many parallel logic operations); restore energy, of peripheral circuitry resets upon restarts. Backup energy is a function of the checkpointing frequency rather than the number of restarts. The dead and restore energy are functions of the number of restarts. For RAT, backup practically has no latency overhead, as backups occur during the execution of each instruction naturally. Dead (restore) latency is only a negligible 0.0000028% (0.0000006%) of the total compute latency, as well.

VII. Limitations

A. Reliability

We assume that FLY and RAT can continuously transfer data back and forth. There are a number of complicating factors which can disrupt proper operation. FLY may begin transmission, but lose power for an extended period of time. In this case, RAT would spend its entire time waiting for an input that may never arrive. RAT can be augmented with a watchdog timer to detect such cases, upon which it can cancel data reception. Additionally, if RAT goes without power for an extended period of time, when it is powered on again, it may work on input which is old and no longer relevant. Keeping beyond edge devices working on relevant tasks over time is an important open problem [40].

In the worse case, either FLY or RAT may never come back. While deploying *fleets* of beyond edge miniservers and devices [11] may help, it may not be practically possible to recover the corresponding data collected by FLY or processed by RAT. Missed events due to lost data as such may have severe implications depending on the specific deployment scenario. However, we should also note that such data losses and missed events are perfectly possible in a fully functional intermittent computing system if critical data samples arrive during power outages.

B. Limitations Induced by the Power Source Capacity

In intermittent computing systems energy efficiency dictates the time to finish computations, as these systems typically spend most of their time powered-off waiting to harvest sufficient energy [14], [35]. A smaller capacity power source translates into a smaller power budget, which incurs more frequent power-downs. As a result the time to finish computations increases.

Homomorphic computation, even without bootstrapping, is energy hungry. Harvesting enough energy beyond the edge poses an additional challenge in this case. Even simple computations may take a long time. Additionally, as solar power is the only power source which reaches the required levels to make the case for RAT in our illustrative example (Fig. 1), beyond edge miniservers such as RAT would struggle to function at night. To facilitate operation at more modest power levels, deploying *fleets* of beyond edge miniservers [11] may help, where each individual miniserver may compute slowly, but their combined effort may reach high-enough throughput.

This paper demonstrates that even relatively small scale representative computations such as SVM inference – feasible on highly power-constrained beyond-edge devices like FLY – benefit from offloading to a beyond-edge miniserver like RAT. While larger workloads may further underscore RAT's advantages, problem sizes and/or computational depth remain constrained by the inherent limitations of beyond-edge environments, despite RAT's relatively higher power budget. We also present a single-node analysis, noting that practical deployments will likely involve coordinated fleets of RAT and FLY. We leave the exploration of such multi-node systems to future work.

VIII. Related Work

Numerous beyond edge devices have been designed to function correctly in intermittent contexts including ML accelerators [14]. This body of work typically use traditional architectures augmented with nearby non-volatile memory for fast backup operations. ResiRCA [31] uses in-memory computing to accelerate parts of ML inference, as well, and adapts the degree of parallelism to match the harvested energy. However, ResiRCA relies on a battery to maintain a controlling CPU. MOUSE represents another intermittent-safe in-memory accelerator for inference [35]. Contrary to RAT, MOUSE maintains standard memory format, which incurs unnecessary space and energy overhead. At the same time, MOUSE can only compute either along columns or along rows, but not in both dimensions which is the case for RAT. Finally, RAT is tightly tailored to homomorphic SVM inference. Mapping homomorphic inference onto MOUSE is possible, however, would incur an excessive time and energy overhead due to frequent read and writes to perform intra-array data transfers, which reduce to mere logic operations in RAT.

Numerous accelerators for homomorphic computing also exist, but typical designs are not tailored for beyond edge operation. Designs based on FPGA platforms such as [27], [39] typically show performance improvements for a limited set of homomorphic operations due to the limited on-chip hardware resources, which often restricts their applicability to narrow use cases. Alternatively, ASIC implementations such as [33] support wider range of operations needed for homomorphic computation. However, they often require larger on-chip memory to mitigate the overhead of frequent off-chip data transfers, necessitating increased area and energy budgets, which can make them less suitable for beyond edge use cases – possible exceptions include [10] which focus on more resource-constrained devices, although not beyond edge. Alternative approaches such as [26] to mitigate the data transfer overhead rely on processing-in-memory. Although effective at minimizing data transfer overheads compared to ASIC implementations, they cannot support intermittent operation in beyond edge environments.

IX. Conclusion

Beyond edge devices solely harvest their energy from the environment, which significantly limits the available power budget for computation as well as communication. This challenges not only direct local processing of the collected data by the beyond edge device, but also the natural alternative of offloading computation to a remote server. Based on the observation that communication overhead fundamentally increases with distance, in this paper, we make the case for light-weight *miniservers*, strategically placed beyond the edge themselves to service beyond edge devices in their immediate vicinity without compromising security. Such miniservers also harvest energy, however, have access to higher capacity power sources than the beyond edge devices they serve, simply due to their geographical placement. Our miniserver design leverages extreme energy efficiency of intermittent-safe nonvolatile processing-in-memory to support homomorphic operation to preserve privacy. We demonstrate that even for relatively small scale representative computations – which are more likely to fit into the tight power budget of beyond edge devices for local processing – deploying such beyond edge miniservers can lead to higher performance.

To this end, we consider representative deployment scenarios of practical importance, including but not limited to agricultural systems or building structures, where RAT enables highly energy-efficient real-time data processing.

References

[1] "Documents for EMD3D256M*08G1-150CBS1*." Accessed: Nov. 17, 2021. [Online]. Available: https://www.everspin.com/supportdocs/EMD3D256M08G1-150CBS1

[2] A. Q. A. Al Badawi, Y. Polyakov, K. M. M. Aung, B. Veeravalli, and K. Rohloff, "Implementation and performance evaluation of RNS variants of the BFV homomorphic encryption scheme," *IEEE Trans. Emerg. Topics Comput.*, vol. 9, no. 2, pp. 941–956, Apr./Jun. 2021.

[3] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, "A public domain dataset for human activity recognition using smartphones," in *Proc. ESANN*, 2013.

[4] S. Arabi, E. Sabir, and H. Elbiaze, "Information-centric networking meets delay tolerant networking: Beyond edge caching," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Piscataway, NJ, USA: IEEE Press, 2018, pp. 1–6.

[5] T. Bouguera, J.-F. Diouris, J.-J. Chaillout, R. Jaouadi, and G. Andrieux, "Energy consumption model for sensor nodes based on lora and lorawan," *Sensors*, vol. 18, no. 7, 2018, Art. no. 2104.

[6] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical GapSVP," in *Annu. Cryptol. Conf.*, Berlin, Germany: Springer, 2012, pp. 868–886.

[7] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Trans. Intell. Syst. Technol. (TIST)*, vol. 2, no. 3, 2011, Art. no. 27.

[8] Z. Chowdhury et al., "Efficient in-memory processing using spintronics," *IEEE Computer Architecture Letters*, vol. 17, no. 1, pp. 42–46, Jan./Jun. 2017.

[9] A. Colin, E. Ruppel, and B. Lucia, "A reconfigurable energy storage architecture for energy-harvesting devices," in *Proc. 23rd Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, 2018, pp. 767–781.

[10] S. Das, M. V. D. Hagen, S. Patil, C. Erbagci, B. Lucia, and K. Mai, "A 10.33 uj/encryption homomorphic encryption engine in 28nm CMOS with 4096-degree 109-bit polynomials for resource-constrained IoT clients," in *Proc. IEEE 49th Eur. Solid State Circuits Conf. (ESSCIRC)*, 2023, pp. 193–196.

[11] B. Denby and B. Lucia, "Orbital edge computing: Nanosatellite constellations as a new class of computer system," in *Proc. 25th Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, 2020, pp. 939–954.

[12] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "NVSim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 31, no. 7, pp. 994–1007, Jul. 2012.

[13] G. Gobieski, et al., "Riptide: A programmable, energy-minimal dataflow compiler and architecture," in *Proc. 55th IEEE/ACM Int. Symp. Microarchit. (MICRO)*, Piscataway, NJ, USA: IEEE Press, 2022, pp. 546–564.

[14] G. Gobieski, B. Lucia, and N. Beckmann, "Intelligence beyond the edge: Inference on intermittent embedded systems," in *Proc. 24th Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, 2019, pp. 199–213.

[15] M. A. Green, Y. Hishikawa, E. D. Dunlop, D. H. Levi, J. Hohl-Ebinger, and A. W. Ho-Baillie, "Solar cell efficiency tables (version 52)," *Prog. Photovolt. Res. Appl.*, vol. 26, no. 7, pp. 427–436, 2018.

[16] H. Greenspan, B. Van Ginneken, and R. M. Summers, "Guest editorial deep learning in medical imaging: Overview and future promise of an exciting new technique," *IEEE Trans. Med. Imag.*, vol. 35, no. 5, pp. 1153–1159, May 2016.

[17] R. Harjani and S. Chaubey, "A unified framework for capacitive series-parallel DC-DC converter design," in *Proc. IEEE Custom Integr. Circuits Conf.*, Piscataway, NJ, USA: IEEE Press, 2014, pp. 1–8.

[18] S. Kim et al., "Ambient RF energy-harvesting technologies for self-sustainable standalone wireless sensor platforms," *Proc. IEEE*, vol. 102, no. 11, pp. 1649–1666, Nov. 2014.

[19] R. Kohavi, "Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid," in *Proc. KDD*, vol. 96, 1996, pp. 202–207.

[20] Y. LeCun et al., "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[21] H. Li et al., "Memristive crossbar arrays for storage and computing applications," *Adv. Intell. Syst.*, vol. 3, no. 9, Sep. 2021, Art. no. 2100017.

[22] C. Liu, Y. Zhang, and H. Zhou, "A comprehensive study of bluetooth low energy," *J. Phys. Conf. Ser.*, vol. 2093, no. 1, 2021, Art. no. 012021.

[23] B. Lucia, B. Denby, Z. Manchester, H. Desai, E. Ruppel, and A. Colin, "Computational nanosatellite constellations: Opportunities and challenges," *GetMobile: Mobile Comput. Commun.*, vol. 25, no. 1, pp. 16–23, 2021.

[24] M. Manic, K. Amarasinghe, J. J. Rodriguez-Andina, and C. Rieger, "Intelligent buildings of the future: Cyberaware, deep learning powered, and human interacting," *IEEE Ind. Electron. Mag.*, vol. 10, no. 4, pp. 32–49, Dec. 2016.

[25] A. Mondal et al., "In situ stochastic training of mtj crossbars with machine learning algorithms," *ACM JETC*, vol. 15, no. 2, 2019.

[26] H. Nejatollahi, S. Gupta, M. Imani, T. S. Rosing, R. Cammarota, and N. Dutt, "Cryptopim: In-memory acceleration for lattice-based cryptographic hardware," in *Proc. 57th ACM/IEEE Des. Autom. Conf. (DAC)*, Piscataway, NJ, USA: IEEE Press, 2020, pp. 1–6.

[27] H. Nejatollahi, S. Shahhosseini, R. Cammarota, and N. Dutt, "Exploring energy efficient quantum-resistant signal processing using array processors," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Piscataway, NJ, USA: IEEE Press, 2020, pp. 1539–1543.

[28] W. S. Noble, "What is a support vector machine?" *Nature Biotechnol.*, vol. 24, no. 12, pp. 1565–1567, 2006.

[29] Ö. Özerk, C. Elgezen, A. C. Mert, E. Öztürk, and E. Savaş, "Efficient number theoretic transform implementation on gpu for homomorphic encryption," *J. SuperComput.*, vol. 78, pp. 1–33, 2021.

[30] S. Park, J. Byun, J. Lee, J. H. Cheon, and J. Lee, "He-friendly algorithm for privacy-preserving SVM training," *IEEE Access*, vol. 8, pp. 57414–57425, 2020.

[31] K. Qiu et al., "Resirca: A resilient energy harvesting reram crossbar-based accelerator for intelligent embedded processors," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Piscataway, NJ, USA: IEEE Press, 2020, pp. 315–327.

[32] P. Rajak, A. Ganguly, S. Adhikary, and S. Bhattacharya, "Internet of Things and smart sensors in agriculture: Scopes and challenges," *J. Agriculture Food Res.*, vol. 14, Dec. 2023, Art. no. 100776.

[33] B. Reagen et al., "Cheetah: Optimizing and accelerating homomorphic encryption for private inference," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit. (HPCA)*, Piscataway, NJ, USA: IEEE Press, 2021, pp. 26–39.

[34] S. Resch et al., "PimCity: A compute in memory substrate featuring both row and column parallel computing," in *Proc. IEEE Int. Conf. Rebooting Comput. (ICRC)*, Piscataway, NJ, USA: IEEE Press, 2023, pp. 1–10.

[35] S. Resch et al., "Mouse: Inference in non-volatile memory for energy harvesting applications," in *Proc. 53rd Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, Piscataway, NJ, USA: IEEE Press, 2020, pp. 400–414.

[36] J. Rosenthal and M. S. Reynolds, "A 158 PJ/bit 1.0 Mbps bluetooth low energy (BLE) compatible backscatter communication system for wireless sensing," in *Proc. IEEE Topical Conf. Wireless Sensors Sensor Netw. (WiSNet)*, Piscataway, NJ, USA: IEEE Press, 2019, pp. 1–3.

[37] J. San Miguel et al., "The EH model: Early design space exploration of intermittent processor architectures," in *Proc. 51st Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, Piscataway, NJ, USA: IEEE Press, 2018, pp. 600–612.

[38] "Microsoft SEAL (release 3.7)," Microsoft Res., Redmond, WA, USA, Sep. 2021. [Online]. Available: https://github.com/Microsoft/SEAL

[39] S. Sinha Roy, F. Turan, K. Jarvinen, F. Vercauteren, and I. Verbauwhede, "FPGA-based high-performance parallel architecture for homomorphic computing on encrypted data," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, 2019, pp. 387–398.

[40] M. Surbatovich, L. Jia, and B. Lucia, “Automatically enforcing fresh and consistent inputs in intermittent systems,” in *Proc. 42nd ACM SIGPLAN Int. Conf. Program. Lang. Des. Implementation*, 2021, pp. 851–866.

[41] F. Turan, S. S. Roy, and I. Verbauwhede, “HEAWS: An accelerator for homomorphic encryption on the amazon AWS FPGA,” *IEEE Trans. Comput.*, vol. 69, no. 8, pp. 1185–1196, Aug. 2020.

[42] M. van der Hagen and B. Lucia, “Client-optimized algorithms and acceleration for encrypted compute offloading,” in *Proc. 27th ACM Int. Conf. Archit. Support Program. Lang. Oper. Syst. (ASPLOS)*, New York, NY, USA: ACM, 2022, pp. 683–696.

[43] M. Zabihi et al., “Analyzing the effects of interconnect parasitics in the STT CRAM in-memory computational platform,” *IEEE J. Explor. Solid-State Computat.*, vol. 6, no. 1, pp. 71–79, Jun. 2020.

[44] M. Zabihi et al., “Using spin-hall mtjs to build an energy-efficient in-memory computation platform,” in *Proc. 20th Int. Symp. Qual. Electron. Des. (ISQED)*, Piscataway, NJ, USA: IEEE Press, 2019, pp. 52–57.

[45] Y. Zhang, “The future of mobile edge computing,” in *Mobile Edge Computing*, Cham, Switzerland: Springer Int. Publishing, 2022, pp. 81–105.

Salonik Resch received the M.S. and Ph.D. degrees in electrical engineering from the University of Minnesota. His research interests include quantum computing, processing in memory, and intermittent computing.

Hüsrev Cılasun received the Ph.D. degree from the University of Minnesota. His research interests include computer architecture, in-memory computing, FPGA/ASIC RTL design, signal processing, and video coding.

Zamshed I. Chowdhury received the B.Sc. and M.S. degrees in applied physics, electronics and communication engineering from the University of Dhaka, Bangladesh, and the Ph.D. degree from the Department of Electrical and Computer Engineering, University of Minnesota, Twin Cities, USA. Prior to this, he was a Faculty Member with Jahangirnagar University, Bangladesh. His research interests include hardware accelerator design, in-memory computing, and approximate computing.

Masoud Zabihi received the B.Sc. degree in 2010, from the University of Tabriz, and the M.S. degree in 2013, from Sharif University of Technology, both in electrical engineering and electronics, and the Ph.D. degree in electrical engineering from the University of Minnesota. His research interests include in-memory computing and spintronic-based memory technologies.

Yang Lv received the Ph.D. degree in electrical engineering from the University of Minnesota, Twin Cities. Currently, he is a Research Associate with the Professor Jian-Ping Wang’s Group, University of Minnesota. His research interests span spintronic and magnetic materials, novel spintronic device physics and phenomena, and unconventional computing enabled by spintronic devices.

Jian-Ping Wang (Fellow, IEEE) is currently the Robert F. Hartmann Chair and a Distinguished McKnight University Professor of electrical and computer engineering with the University of Minnesota. He is the Director of the Center for Spintronic Materials for Advanced Information Technology (SMART), one of two SRC/NIST nCORE research centers. He was the Director of the Center for Spintronic Materials, Interfaces and Novel Architectures (C-SPIN). CSPIN was one of six SRC/DARPA STARnet program centers. His inventions have been used in both HDD products and STT-RAM products. He was the recipient of the Information Storage Industry Consortium (INSIC) Technical Award in 2006 for his pioneering work in exchange coupled composite magnetic media. He was the recipient of the 2019 SRC Technical Excellence Award for his innovations and discoveries in nanomagnetics and novel materials that accelerated the production of magnetic random-access memories. He is an APS fellow.

Sachin S. Sapatnekar (Fellow, IEEE) received the B.Tech. degree from the Indian Institute of Technology, Bombay, the M.S. degree from Syracuse University, and the Ph.D. degree from the University of Illinois. He taught at Iowa State University, from 1992 to 1997, and has been with the University of Minnesota, since 1997, where he holds the Distinguished McKnight University Professorship and the Robert and Marjorie Henle Chair. He is the recipient of nine conference a Best Paper Awards, a Best Poster Award, a two ICCAD ten-year Retrospective Most Influential Paper Awards, the SRC Technical Excellence Award, and the SIA University Researcher Award. He is also a fellow of ACM.

Ismail Akturk (Member, IEEE) received the Ph.D. degree from the University of Minnesota, Twin Cities. Currently, he is an Assistant Professor with the Computer Science Department, Özyeğin University, Istanbul, Turkey. His research interests include improving energy efficiency, scalability and fault tolerance of computing systems, and emerging and nonconventional computing paradigms.

Ulya R. Karpuzcu (Member, IEEE) received the M.S. and Ph.D. degrees in electrical and computer engineering from the University of Illinois, Urbana-Champaign. Currently, she is a Jim and Sara Anderson Professor with the Department of Electrical and Computer Engineering, University of Minnesota. Her research interests span physics of computing, Ising machines, computing-in-memory, intermittent computing, hardware security, reliability, sustainability, and energy-efficiency.