HADES: A Hardware-Resident Substrate for Energy Admission in Intermittent Computing

Han Wang[1], Chong Zhang[2], Qianhe Meng[1], Yizhe Zhao[1], Ruizhe Zhang[1], and Li Lu[1,✉]
[1]University of Electronic Science and Technology of China (UESTC), Chengdu, China
[2]Southwest Petroleum University (SWPU), Chengdu, China
{wang_han, qianhe, zhaoyize, zhang.ruizhe}@std.uestc.edu.cn; zhangchong92@swpu.edu.cn; luli2009@uestc.edu.cn

***Abstract*—Batteryless devices operate from ambient energy and avoid the maintenance burden of replaceable batteries. Their power supply is usually intermittent, so a bounded task should begin only after local storage can support its execution. Existing systems usually make this decision in software. The microcontroller unit (MCU) wakes, senses the storage state, and runs admission code before useful work starts. Under weak input, this path can consume up to 70% of the incoming power budget.**

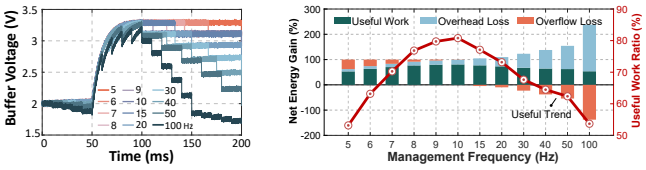
This paper presents HADES, an always-on mixed-signal substrate that performs energy admission while the MCU remains off. Its Energy Intake Front-End selects source–bank paths from characterized loaded-voltage windows. An in situ Energy Aggregator produces a calibrated analog proxy for deliverable energy across distributed capacitor banks. A Dispatch Controller applies the selected task threshold, releases the MCU, and holds the execution grant through the task window. Using a 180 nm process design kit (PDK), we calibrate a co-simulation that integrates analog/mixed-signal (AMS) models with register-transfer-level (RTL) control. In this flow, HADES reduces modeled runtime admission energy by more than 98% relative to the software-mediated baseline. The reported admission-control core occupies 0.29% of the evaluated system-on-chip (SoC) area. Across 12 workloads, HADES completes 2.4× more tasks per joule and achieves up to 2.5× higher throughput than the baseline.

***Index Terms*—intermittent computing, energy admission, energy-harvesting systems, mixed-signal architecture**

I. INTRODUCTION

The ever-growing Internet of Things (IoT) is pushing computation into increasingly diverse and hard-to-reach environments, from orbital platforms to underwater infrastructure [1]–[3]. In these settings, batteries remain a major deployment bottleneck: they add bulk, limit lifetime, increase maintenance cost, and impose an environmental burden. Batteryless devices offer a compelling alternative by harvesting energy from ambient sources such as light, radio frequency (RF), thermal gradients, and motion, thereby enabling long-lived, maintenance-free operation. Their defining constraint, however, is that computation must proceed under highly intermittent power. A device must slowly accumulate charge in an energy buffer, expend it in a brief burst of execution, and then repeat this Sisyphean cycle as energy becomes available. If a task is admitted at the wrong time, power can fail mid-execution, destroying volatile state and compromising forward progress. This challenge has made intermittent computing (ImC) a key systems abstraction for batteryless platforms [4], [5].

✉ Corresponding author: Li Lu.



(a) Transient voltage. (b) The efficiency paradox.

Fig. 1. Why software-resident admission is costly in ImC.

In most ImC systems today, energy admission remains software-resident. The MCU wakes, samples the energy buffer, interprets the result, and decides whether to execute, checkpoint, or wait. The control path that protects forward progress draws from the same scarce energy budget as useful work, which makes software-resident admission costly under weak input. The paradox is illustrated in Fig. 1. At the circuit level (Fig. 1(a)), more frequent monitoring can reduce overflow and improve responsiveness, but it also repeatedly pulls the MCU out of deep sleep and drains accumulated charge. At the system level (Fig. 1(b)), energy spent on interrupts, ADC sampling, and runtime bookkeeping directly reduces the energy available for useful application execution.

This tension becomes most severe exactly when the system is weakest. Under erratic harvesting conditions, the runtime must monitor energy more aggressively to avoid brownout, precisely when the platform can least afford the added overhead. Prior silicon measurements show that voltage sensing alone can consume 27%–71% of harvested energy in these regimes [6]. Once hardware wakeups, control-flow disruption, and firmware-level decision logic are included, the admission loop becomes a structural liability and loses its intended role as a safeguard. The MCU spends a non-trivial fraction of its budget deciding whether it may run.

One response is to offload pieces of this path into hardware. Prior work has introduced voltage supervisors [6], [7], reconfigurable capacitor banks [8], [9], and hardware timing support [10] to reduce parts of the runtime overhead. Yet software typically remains on the critical path. Hardware raises an event, the MCU wakes, firmware interprets the energy state, and the runtime decides whether execution may proceed. This residual dependence on software becomes increasingly problematic as batteryless platforms evolve toward multi-source harvesting [11], [12] and reconfigurable storage fabrics [8], [9]. In such systems, usable energy is physically

distributed and configuration-dependent. A single-node voltage is no longer a faithful summary of execution readiness. Energy admission therefore extends beyond a cheaper intermittency check. It is a distributed energy-reduction problem that must itself be solved under extreme energy constraints.

HADES moves *energy admission* out of the software path. The mixed-signal substrate evaluates stored energy and task requirements while the MCU remains off, then opens a bounded execution window when the profiled conditions are met. Application execution and ordinary interrupt behavior remain unchanged after release, except for a dedicated completion write at the task boundary. The recurring energy-sensing and scheduling loop no longer occupies the MCU instruction stream. This design raises three coupled challenges.

Challenge 1: Efficient energy intake under heterogeneous sources. Practical harvesters differ in voltage, impedance, and temporal behavior. Their useful operating regions also move with illumination, distance, temperature, and motion [11], [12]. A separate high-precision tracking loop for every source carries a substantial sensing and control cost at microwatt power levels. HADES uses source-specific conditioning and characterized loaded-voltage windows to select a compatible storage bank. The front end evaluates these windows with analog comparators and changes source–bank paths through a small routing controller.

Challenge 2: Interpreting usable energy in a distributed storage fabric. A reconfigurable capacitor fabric may hold charge at several voltages. A local threshold can miss energy that remains usable through the shared load path. HADES reduces the bank state in analog hardware. Capacitance-weighted branches sense the physical bank terminals and generate a single admission voltage. The resulting signal is calibrated against deliverable energy over the supported bank configurations. It serves as a one-sided admission proxy for task thresholds and removes per-bank ADC scans and firmware arithmetic from the decision path.

Challenge 3: Releasing a bounded task without runtime admission code. The admission result must be connected to a concrete task and a controlled power-up sequence. Task requirements are profiled at an edge tier and encoded as threshold, entry, and timing metadata. On device, the Dispatch Controller selects a descriptor, waits for the analog reference and routing fabric to settle, and then enables the load supply, clock, and reset in sequence. An execution-window latch preserves the grant through normal load-induced droop, while the task closes the window through the dedicated completion endpoint. Program-state persistence, recovery after an unexpected failure, and idempotent I/O remain responsibilities of the intermittent runtime used by the application.

These components form a hardware admission pipeline. The Energy Intake Front-End improves energy delivery into storage. The Energy Aggregator reduces distributed storage bank state to a calibrated voltage-domain proxy. The Dispatch Controller binds that state to a task and controls the execution window. We implement HADES in a unified mixed-signal SoC co-simulation flow. The analog circuits are modeled in Verilog-A with transistor-level calibration. The controller and an open-MSP430 core are implemented in synthesizable SystemVerilog and evaluated with post-synthesis power estimates in a 180 nm process design kit (PDK). Across 12 benchmarks and real-world energy traces, HADES reduces runtime admission overhead and improves end-to-end throughput under the evaluated source, storage, and task profiles.

The contributions of this work are as follows:

- We present an always-on mixed-signal substrate that performs task-energy admission outside the MCU instruction stream.
- We design a capacitance-weighted Energy Aggregator and a one-sided calibration method that convert distributed bank state into a hardware-comparable proxy for deliverable energy.
- We develop a source–bank intake front end and a Dispatch Controller that connect characterized energy conditions to bounded task release, completion, and power sequencing.

II. Background and Related Work

A. The Software-Mediated ImC Stack

Fig. 2 illustrates the conventional architecture underlying most intermittent-computing systems. The stack is physically and logically partitioned into three tiers: energy harvesting and storage hardware, the software-defined ImC management layer, and the application-oriented payload. While this model successfully enables batteryless operation, its execution flow exposes why energy admission remains a fundamental bottleneck.

At the hardware level (Fig. 2, top), erratic ambient energy (e.g., solar, RF, thermal, kinetic) is conditioned by a voltage regulator and buffered in a capacitor bank. Because raw ambient power cannot reliably sustain continuous execution, a hardware supervisor (such as a voltage comparator) acts as a coarse-grained trigger. When the storage voltage (V_{cap}) crosses a predefined threshold, the supervisor fires a hardware interrupt to wake the microcontroller (MCU) from deep sleep.

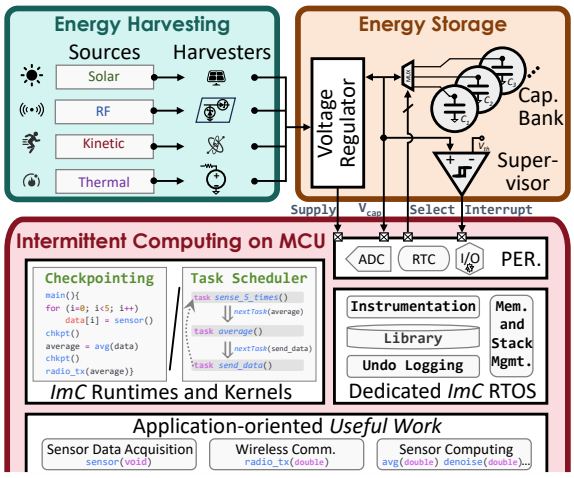


Fig. 2. A typical software-mediated ImC stack.

Once awakened, the MCU enters the software-mediated ImC layer (Fig. 2, middle). Before executing the application, the processor must first navigate a complex suite of runtimes and kernels. It typically samples the exact capacitor voltage via an analog-to-digital converter (ADC), evaluates task dependency graphs within the task scheduler, and orchestrates state preservation mechanisms such as software checkpointing (e.g., inserting `chkpt()` routines) or undo logging [13]–[15]. Only after this heavy runtime management concludes is the MCU permitted to drop down to the bottom layer to perform actual useful work, such as sensor data acquisition, signal processing, or wireless communication.

We can see from this stack that *energy management* and *useful work* are inextricably coupled within the same execution pipeline. Both draw from the same scarce energy pool and compete for the same CPU cycles. Energy admission thus becomes an expensive software service embedded within the very instruction stream it attempts to regulate. This highlights an important distinction between two challenges unique in ImC: *state persistence* (preserving volatile state across power failures) and *energy admission* (determining when sufficient energy exists to safely begin the next bounded execution window). While prior work has rigorously addressed persistence—represented by the logging and checkpointing blocks in Fig. 2—admission is still overwhelmingly realized as a power-hungry software burden.

B. Related Work

Software-Mediated Admission. Most intermittent-computing systems place admission in software. Early systems such as Mementos [16] and Dewdrop [13] established software-managed recovery and energy-aware execution for intermittently powered devices. Subsequent work refined this model through task decomposition, checkpoint reduction, and adaptive scheduling, including Alpaca [17], Chain [18], and related compiler/runtime techniques [15], [19]–[21]. More recent systems extend these ideas to peripheral-rich, time-sensitive, and multithreaded workloads [14], [22]–[27]. These works substantially improve correctness and utilization under intermittency, but the admission path remains software-mediated: the MCU still wakes, inspects energy state, and decides whether execution may begin.

Hardware-Assisted Admission. Prior work has already moved important parts of energy management into hardware, but the admission decision often still passes through software. Failure Sentinels [6] reduce the cost of detecting unsafe energy conditions, while hardware-assisted designs add monitoring, isolation, supervision, and timing support [10], [28], [29]. Reconfigurable charge-storage systems [8], [9], [30], [31] improve how energy is buffered and reclaimed. Yet in these systems, hardware typically reports power-system state, and firmware turns that report into an execution decision. This boundary is costly in intermittent regimes. A short energy opportunity may first be spent waking digital logic, reconstructing fragmented storage state, and running management code before any application instruction executes. HADES removes that handoff by placing source–storage matching, parallel bank-state reduction, threshold comparison, and execution-window enforcement in one always-on admission substrate. The MCU is released only after admission has already been decided in hardware. Short harvested-energy windows can then reach application execution with less MCU-side management energy.

Post-Admission Execution and Recovery Optimizations. A separate line of work assumes that a compute window already exists and optimizes execution, persistence, and recovery within or across it. Prior proposals improve nonvolatile-memory (NVM) behavior, cache efficiency, and recovery overhead through designs such as Clank [32], ReplayCache [33], NvMR [34], Write-light Cache [35], SweepCache [36], LightWSP [37], and WarmCache [38], along with related intermittent cache and data-management techniques [39]–[41]. Other work improves recovery efficiency or extends intermittent execution to multicore and hybrid persistence settings [42]–[46]. These techniques are complementary to HADES: they optimize what happens once execution has been admitted, whereas we target the admission path itself.

Pre-Admission Energy Intake and Buffering Optimizations. Orthogonal work improves the amount of energy available for future execution through multi-source harvesting, power-management integrated circuits (PMICs), and maximum-power-point-tracking (MPPT)-style control loops [11], [12], [47]–[50]. These methods improve source-side energy capture, while computation release under distributed storage remains a separate admission problem. At the microwatt scale, the control loop must also fit within the harvested-power budget.

Where HADES Fits. HADES differs in one key respect: it makes admission itself hardware-resident. HADES extends hardware support beyond firmware wakeup and performs energy intake, direct analog reduction of the calibrated bank-state proxy, and task release inside an always-on mixed-signal substrate. It complements prior work on persistence, post-admission execution, and harvest-side optimization, while changing where admission is enforced.

III. MAXIMIZING ENERGY HARVESTING EFFICIENCY

For a batteryless system, useful throughput depends on both sides of the energy path: how much ambient energy reaches storage and how effectively that energy is later converted into completed computation. These two stages share the same limited budget. A harvesting controller may improve source utilization, yet frequent MCU wakeups, ADC conversions, and iterative regulation can consume a substantial part of the additional energy it recovers.

HADES therefore treats harvesting as the *income side* of end-to-end energy management. The design objective is net energy delivered into storage after accounting for the cost of the control path. To this end, the Energy Intake Front-End replaces continuous digital tracking with source–storage matching over a reconfigurable capacitor bank. It identifies favorable source–bank pairs using voltage-window comparisons,

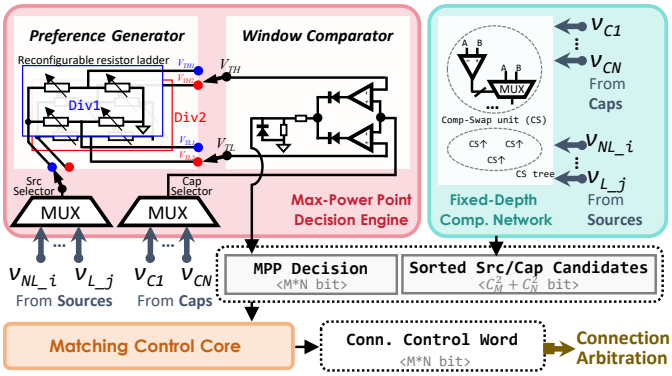


Fig. 3. **Energy Intake Front-End.** The Preference Generator produces source-specific voltage windows; the Window Comparator identifies feasible source–bank pairs; the fixed-depth comparison network orders the candidates; and the Matching Control Core forwards the resulting control word to Connection Arbitration.

orders the resulting candidates with a fixed-depth network, and commits a valid connection through lightweight topology control.

Energy Intake Front-End. Conventional harvesting systems commonly use maximum-power-point tracking (MPPT) to monitor source conditions and iteratively adjust the effective load [51]. This approach is effective when the tracking circuitry consumes only a small fraction of the available power. At microwatt scale, however, source sensing, controller wakeup, and repeated feedback adjustment can approach the energy gained from tighter tracking. The relevant metric consequently becomes the energy retained by the storage system rather than tracking accuracy alone.

Heterogeneous harvesting further complicates this tradeoff. Solar, RF, thermal, and kinetic sources differ in voltage range, effective impedance, startup behavior, and temporal variation. Connecting them to a common passive node, as in diode OR-ing [49], can force one or more sources away from a productive operating point. Equipping every source with an independent active MPPT loop [48] avoids this coupling, but replicates sensing, regulation, and control state as the number of inputs grows.

The key observation behind HADES is that a multi-bank capacitor fabric already exposes several candidate load voltages. Rather than continuously synthesizing a new operating point for each harvester, the front end selects the bank whose present voltage is compatible with the source. Harvest-side regulation is thereby reduced to choosing among source–bank combinations already available in the system.

Fractional open-circuit-voltage behavior provides a useful starting point for identifying these operating regions. Approximately linear sources, such as thermoelectric and piezoelectric harvesters, often reach peak power near $0.5V_{OC}$, whereas photovoltaic cells and RF rectennas commonly operate near a source-dependent fraction of V_{OC}, frequently around 0.7–0.8 [50]. The deployed matching limits, however, are obtained from characterization of the complete source interface rather than directly applying these fractions at runtime.

Each raw harvester first passes through its source-appropriate rectifier or conditioning stage. We denote the resulting loaded source voltage by $V_{H,j}$. During platform characterization, source j is swept over its expected input and load range. The sweep selects two source-specific ladder ratios, $k_{TL,j}$ and $k_{TH,j}$, from the available resistor-ladder taps. At runtime, these ratios generate $V_{TL,j} = k_{TL,j}V_{H,j}$ and $V_{TH,j} = k_{TH,j}V_{H,j}$. The selected taps delimit a productive source–bank operating region over the characterized range while accounting for conditioner behavior, switch-path loss, and the voltage headroom required for forward charge transfer.

The Source MUX and Cap. MUX in Fig. 3 enumerate the supported source–bank pairs. For each pair, the Preference Generator produces the corresponding limits and the Window Comparator records whether capacitor voltage $V_{C,i}$ lies within them. Across M sources and N capacitor banks, the accumulated candidate set is

$$\Phi = \{(i,j) \mid k_{TL,j}V_{H,j} \le V_{C,i} \le k_{TH,j}V_{H,j}\}.$$

An asserted comparator result therefore identifies a bank that both preserves the required charging direction and places the source in a productive operating region. The comparison remains entirely in the voltage domain and requires neither per-source digitization nor MCU-side evaluation.

The resulting bitmap can contain several candidates. A source may be compatible with more than one bank, while multiple sources may request the same bank. The fixed-depth compare–swap network shown in Fig. 3 orders the source and capacitor candidates according to the voltage ordering and source-class priority already encoded by the selector paths. The Matching Control Core combines this ordered list with the feasibility bitmap and forwards compatible requests to the arbitration stage. It does not calculate runtime weights or solve a general assignment problem.

The selector scan and comparison depth are fixed by the supported numbers of source and bank ports. Decision latency is therefore bounded at design time and remains independent of MCU execution or application behavior. Variations in illumination, RF coupling, temperature gradient, or motion change the comparator outcomes without changing the structure of the control path.

The voltage windows also reduce unnecessary topology changes. Harvester power commonly remains close to its maximum over a finite voltage interval, so reacting to every small fluctuation yields little additional energy while increasing switch loss, charge redistribution, and control activity. HADES retains the current connection while the selected bank remains within the characterized region. Source ripple, comparator offset, and gradual bank-voltage movement can therefore be absorbed without reconfiguration. A new path is considered when the current pair leaves that region or when a higher-priority feasible candidate becomes available. The front end thus follows meaningful changes in source conditions while keeping repeated digital regulation off the harvested-energy path.

Connection Arbitration. A favorable source–bank pair is not

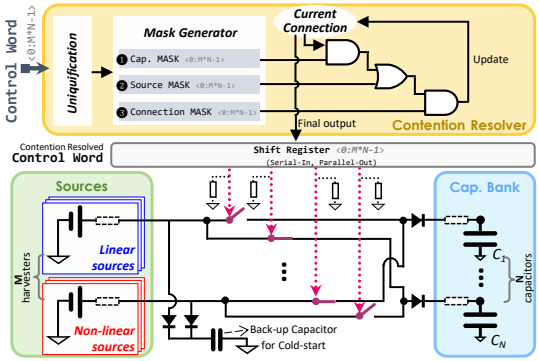


Fig. 4. **Connection Arbitration.** The candidate control word is uniquified and filtered by capacitor, source, and connection masks. The Contention Resolver combines these masks with the current topology, and the shift register commits the resulting source–bank configuration. A backup capacitor supports cold start.

yet a valid physical topology. Several candidates may contend for the same capacitor, and one source may appear in multiple feasible pairs. Applying these requests directly to the switch fabric could connect incompatible paths or expose the storage system to transient intermediate configurations. The matching result therefore passes through the Connection Arbitration stage shown in Fig. 4 before any switch state is committed.

The incoming control word first enters *Uniquification*, which converts repeated requests into an ordered candidate representation. The Mask Generator then derives the three constraints shown in the figure. The `Cap. MASK` removes candidates that contend for the same capacitor. The `Source MASK` restricts each source to one active destination. The `Connection MASK` incorporates the currently committed topology and excludes paths that conflict with an existing connection.

The Contention Resolver combines these masks with the *Current Connection* state and produces the next valid source–bank configuration. The resolved word is shifted into the serial-in/parallel-out register while the current connection remains unchanged. After the complete word has arrived, the existing *Update* control transfers the parallel outputs to the switch fabric as one committed configuration. Candidate evaluation can therefore continue in front of the register without exposing partially shifted states to the capacitor fabric. The admission comparison is held during this update and resumes after the capacitor terminals have settled.

Cold start is handled through the backup capacitor shown at the bottom of Fig. 4. An available source first charges this capacitor through the default startup path, providing the initial energy required by the preference, comparison, and arbitration logic. Once the front end becomes active, normal source–bank matching takes control of the storage fabric.

Together, source–storage matching and connection arbitration establish the income-side path of HADES. The matching stage uses the voltage diversity already present in the capacitor bank to improve source utilization, while the arbitration stage converts candidate pairs into a stable physical topology. Both stages operate outside the MCU instruction stream, allowing the energy-management machinery to remain small relative to the energy it helps recover.

IV. Minimizing Energy Management Spending

Bringing more energy into storage solves only half of the problem. A software-mediated intermittent system can spend a surprising fraction of that energy deciding whether it is allowed to run. The MCU wakes, samples the storage state, interprets the result, selects a task, and either begins useful work or returns to sleep. An unsuccessful attempt still pays the wakeup, sensing, and control-flow cost. As input power falls, this cost changes little while the energy opportunity becomes smaller, and management gradually becomes a first-order workload [8], [21], [30].

Reducing the polling rate lowers overhead, but it also delays response to short energy opportunities and provides only a partial view of a distributed capacitor bank. HADES instead moves the recurring admission path into a compact always-on mixed-signal substrate. The MCU becomes a switched compute domain. It receives power, clock, and reset release after task policy and physical energy state have already produced a grant.

The substrate follows a task request through three components. The **Descriptor Ingress Path** and **Task Table** hold edge-generated policy. The **Energy Aggregator** reduces the distributed capacitor state to one task-comparable signal. The **Dispatch Controller** selects a task, installs its threshold, and controls the lifetime of the execution window. These components remove energy admission from the payload instruction stream while leaving application execution on the MCU.

A. Task Energy Profiling at the Edge

The first question is what the hardware should compare against. A storage voltage has little meaning by itself: a radio transmission, a sensor operation, and a compute kernel may draw similar average power yet differ in startup cost, duration, peak current, and peripheral behavior. Estimating those quantities on the batteryless node would place profiling on the same critical path that admission is meant to protect.

HADES therefore characterizes bounded task instances at the edge. The expensive operation—measuring and bounding task demand—is performed when the task image or platform configuration changes. The frequent operation—deciding whether the task can run now—is reduced to a stored threshold and one hardware comparison.

A profile binds a task binary to a specific execution domain. The binding includes clock setting, active peripheral set, memory mode, storage organization, and the bounded input or retry behavior used during characterization. The same code can have a different energy requirement when any of these conditions changes. A descriptor is consequently reused only while the task and platform remain inside the profiled domain.

1) Task Energy Isolation: For task t, the edge computes an *isolation budget* $E_{\mathrm{iso},t}$ that covers the complete admitted window. Profiling follows the task through startup, MCU execution, memory activity, and peripheral service using the

post-synthesis characterized power states associated with each phase. It also includes loss along the storage-to-load path and the always-on admission circuitry that remains active while the task runs.

Let $P_{\mathrm{task},t}(\tau)$ denote the characterized power of the active MCU, memory, clock, and peripheral states at time τ, and let $P_{\mathrm{path},t}(\tau)$ collect regulator, switch, and interconnect loss at the same storage-side boundary. The baseline demand is

$$E_{\mathrm{base},t} = E_{\mathrm{start},t} + \int_0^{T_{\mathrm{exec},t}} \left(P_{\mathrm{task},t}(\tau) + P_{\mathrm{path},t}(\tau)\right) d\tau, \tag{1}$$

where $E_{\mathrm{start},t}$ includes the fixed cost of enabling the task power path, releasing reset, entering the task, and reporting completion. Using one energy boundary prevents regulator and path losses from being counted once in the task model and again in the storage model.

Total energy alone does not capture every failure mode. A bank may contain enough joules and still fall below the MCU operating voltage when a task produces a sharp current step. During characterization, bank states are therefore retained only when the worst-case launch condition, $V_{\mathrm{bus}} - I_{\mathrm{step},t} R_{\mathrm{eq}} \geq V_{\mathrm{min},t}$, is satisfied over the profiled voltage and current range. Within this valid region, the profile includes an additional reserve $E_{\mathrm{droop},t}$ derived from the same current step, storage ESR, and switch-path impedance. The final isolation budget is

$$E_{\mathrm{iso},t} = \delta \left(E_{\mathrm{base},t} + E_{\mathrm{droop},t}\right), \tag{2}$$

where $\delta \geq 1$ covers residual profiling error, PVT variation, and storage aging. Thus, $E_{\mathrm{droop},t}$ adds reserve within the characterized supply envelope; it is not used to admit a state that already violates the launch-voltage requirement.

The result is a conservative certificate for one bounded task, not a prediction of arbitrary future execution. A change in task code, peripheral behavior, clock plan, or storage organization invalidates the old certificate and triggers edge-side regeneration. During normal operation, the device stores and enforces the resulting threshold; it does not repeat the profiling process.

2) Mapping Energy to Reference Threshold (V_{ref}): The isolation budget is expressed in joules, while the circuit in Fig. 6 makes a voltage-domain decision. Platform characterization connects these representations for the profiled task domain $\mathcal{D}_t$, which includes the supported storage organization and voltage/current range. The edge sweeps these bank states through the implemented branch array, TIA, comparator, and reference ladder, and selects

$$V_{\mathrm{ref},t} = \mathcal{F}_{\mathrm{cal}}\left(E_{\mathrm{iso},t}; \mathcal{D}_t\right). \tag{3}$$

For each task, the chosen threshold keeps all accepted comparator states inside the characterized region that satisfies both the isolation budget and the launch-voltage constraint. The mapping is therefore task- and platform-specific rather than a universal conversion from joules to voltage. The device still performs only a threshold lookup and comparison at runtime.

The resulting threshold is stored as a 6-bit code. The code selects one of 64 ordered taps in the configurable resistor ladder and produces the task-specific V_{ref}. The edge chooses the conservative tap after comparator offset, ladder variation, and the intended isolation margin have been included. When the Dispatch Controller selects another task, it loads the corresponding code; no energy arithmetic or calibration loop executes on the MCU.

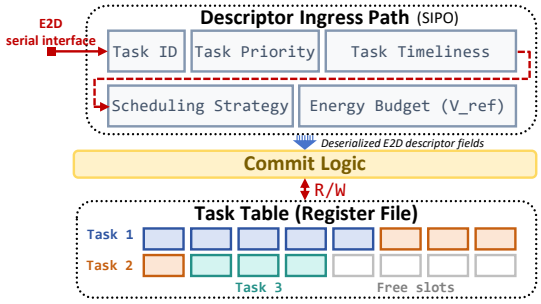


Fig. 5. **On-device E2D descriptor ingress.** The Descriptor Ingress Path deserializes Task ID, Task Priority, Task Timeliness, Scheduling Strategy, and Energy Budget (V_{ref}); Commit Logic writes the complete entry into the Task Table.

3) E2D Descriptor and On-Device Task Ingest: Fig. 5 shows how the edge-generated policy reaches the device. The descriptor contains the fields used by the on-device control path: `Task ID`, `Task Priority`, `Task Timeliness`, `Scheduling Strategy`, and the threshold code represented as `Energy Budget` (V_{ref}). The program itself remains in the task image; the descriptor carries the metadata needed to select and admit it.

The *Descriptor Ingress Path* receives the serial stream and reconstructs the fixed-width fields. After a complete descriptor has arrived, *Commit Logic* writes the fields into one Task Table row. The register-file organization keeps policy available in the always-on domain without waking the MCU. Insert and update operations therefore change future dispatch choices rather than consuming an application execution window.

Each Task Table row ties the threshold to the corresponding task identity. At dispatch, the controller latches the selected `Task ID` in a read-only control register. After reset release, the MCU enters a fixed task trampoline, reads this identifier, and transfers control to the corresponding program entry. The trampoline and identifier read are included in $E_{\mathrm{start},t}$. `Scheduling Strategy` determines whether the descriptor is retired after one execution or remains eligible for repeated execution, while priority and timeliness guide arbitration when several tasks are pending. The edge prepares task policy, the Task Table retains it, and the MCU remains asleep until the policy and physical energy state agree.

B. Energy Aggregation

The next problem is global state. With one capacitor, a local voltage threshold often provides a useful readiness signal. With several isolated banks, the same rule becomes ambiguous. Charge may be distributed across the fabric: no individual

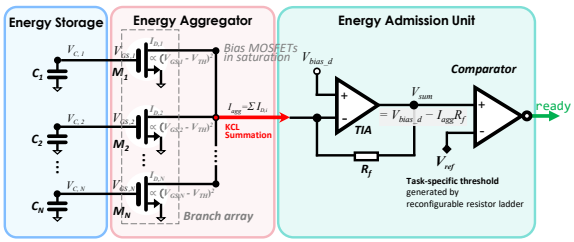


Fig. 6. **Energy Aggregator for distributed capacitor storage.** Capacitor voltages $V_{C,i}$ drive the branch devices M_i; the branch currents meet at the KCL summation node; the TIA produces V_{sum}; and the comparator evaluates V_{sum} against task threshold V_{ref} to generate `ready`.

bank crosses a local threshold, yet the combined state may support the selected task. Reconstructing that state in software requires an ADC scan, per-bank arithmetic, and an MCU wakeup before every execution window.

In the evaluated design, source–bank reconfiguration changes the charging path but does not remove a bank from the common load path. Each physical bank sensed by the branch array therefore remains available to the selected task through the existing isolated discharge path. The task thresholds are calibrated for this same load topology, so the aggregate does not count a bank that the task cannot use.

The Energy Aggregator performs the reduction at the capacitor terminals. Fig. 6 shows the signal path. Capacitor C_i drives branch device M_i. The branch currents meet at the node labeled *KCL Summation* to form I_{agg}. The transimpedance amplifier (TIA) converts the current to V_{sum}, and the comparator evaluates this signal against the task-specific V_{ref}. After reference and routing settling, the qualified `ready` signal is consumed by Grant Logic.

The branch devices use the first-order square-law behavior of a MOS transistor in saturation:

$$I_{D,i} \approx K_i \left[(V_{GS,i} - V_{TH})_+ \right]^2, \tag{4}$$

where $(x)_+ \triangleq \max(x, 0)$. Under the branch bias in Fig. 6, $V_{GS,i}$ follows the corresponding capacitor voltage up to a fixed bias term. Device size sets K_i; for unequal nominal banks, branch width is chosen in proportion to capacitance so that a larger bank contributes more current at the same voltage. This weighting is fixed in the branch geometry and requires no runtime multiplication.

The KCL node and TIA complete the reduction:

$$I_{agg} = \sum_{i=1}^{N} I_{D,i}, \qquad V_{sum} = V_{bias\_d} - I_{agg} R_f. \tag{5}$$

A stronger aggregate bank state produces a larger current and, because the TIA is inverting, a lower V_{sum}. The comparator then forms

$$\texttt{ready} = \begin{cases} 1, & V_{sum} \le V_{ref}, \\ 0, & V_{sum} > V_{ref}. \end{cases} \tag{6}$$

The branch current is an admission coordinate, rather than a closed-form measurement of total capacitor energy for arbitrary voltage combinations. What matters is the final boundary.

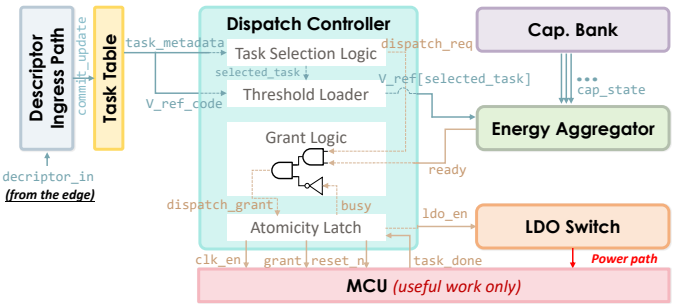


Fig. 7. **Control path of HADES.** Task Selection Logic chooses a Task Table entry, the Threshold Loader applies V_{ref}[`selected_task`], Grant Logic combines `dispatch_req`, `ready`, and `busy`, and the Atomicity Latch drives the MCU power and execution controls until `task_done`.

During platform characterization, balanced and skewed bank states are swept through the transistor-level path, and each task threshold is placed so that accepted states remain inside the task's profiled reserve over the supported operating range. Circuit nonlinearity, branch mismatch, TIA swing, comparator offset, and temperature variation enter this calibration and the corresponding safety evaluation.

C. Dispatch Controller

At this point, the system has a table of task policies and a physical readiness signal. Waking the MCU to combine the two would leave the most frequent part of admission on the payload processor. The Dispatch Controller therefore performs task selection, threshold loading, and execution-window control in the always-on domain.

Fig. 7 follows the path from left to right. The Task Table supplies `task_metadata` and `V_ref_code`. Task Selection Logic chooses `selected_task`. The Threshold Loader applies the corresponding V_{ref} to the Energy Aggregator. Grant Logic combines the resulting `ready` signal with the dispatch request and the current latch state. The Atomicity Latch then controls the lifetime of the MCU domain through `ldo_en`, `clk_en`, `grant`, and `reset_n`.

1) Task Selection and Threshold Loading: Each valid Task Table entry contains the fields shown in Fig. 5. Priority provides the primary order among pending tasks, timeliness refines the choice among tasks of similar priority, and the scheduling strategy controls one-shot or repeated execution. A small fixed-field comparator implements this arbitration; ties are resolved with round-robin state so that repeated selections do not permanently exclude another entry.

Once a candidate is selected, the controller latches its task identity and threshold code. The Threshold Loader programs V_{ref}[`selected_task`] and allows the reference path to settle. The task requests dispatch only after the descriptor is valid, the selected threshold is stable, and any source–bank update has completed. Task identity and threshold therefore remain paired throughout the comparison. If the bank state has not yet reached the selected threshold, the task stays pending and can be reconsidered when more energy arrives.

This ordering has an important consequence. The task threshold is not a global wakeup voltage shared by all work. It belongs to one Task Table entry and changes with the selected candidate. A short, low-cost task can therefore run from an energy state that is insufficient for a longer task, without requiring the MCU to inspect the queue or interpret capacitor voltages.

2) Grant Generation and Atomicity Enforcement: Grant Logic consumes the three signals shown in Fig. 7: the qualified request from Task Selection Logic, the settled admission result from the Energy Aggregator, and the latch feedback indicating whether another task owns the MCU domain. It forms

$$\texttt{dispatch\_grant} = \texttt{dispatch\_req} \wedge \texttt{ready} \wedge \neg \texttt{busy}.$$

Here, `dispatch_req` is asserted only after descriptor, threshold, and routing state are valid. A task therefore begins after its threshold has been applied, the comparator has accepted the settled bank state, and the preceding execution window has closed.

The admission result is momentary, while task execution spans many cycles. This distinction is the reason for the Atomicity Latch. Once the MCU starts drawing current, capacitor voltage naturally falls and V_{sum} may cross back over V_{ref}. That movement reflects consumption of the admitted reserve; it should not withdraw the grant in the middle of the task. The latch records `dispatch_grant`, feeds back `busy` to block overlapping dispatches, and holds the execution window until the task reaches its completion boundary. The latch preserves grant continuity; persistent-state atomicity and idempotent external I/O remain responsibilities of the intermittent runtime.

The latched state drives the signals at the bottom of Fig. 7. `ldo_en` enables the task power path. `clk_en`, `grant`, and `reset_n` release the MCU into the fixed trampoline, which reads the latched Task ID and enters the selected task. From the processor's perspective, the useful task then follows its ordinary instruction path; the right to run has already been established outside that path.

The task reports completion through `task_done`. This signal marks the task boundary rather than every application interrupt. Sensor, timer, and radio handlers remain part of ordinary execution inside the open window. `task_done` is the normal close condition; the existing reset or brownout path also clears the latch if the execution window ends unexpectedly. The intermittent runtime then handles any required recovery. After a normal completion, the scheduling strategy determines whether the descriptor is retired or remains eligible for another iteration.

The complete admission path can now be read across Fig.s 5–7. The edge turns a bounded task into a threshold. The Descriptor Ingress Path places that policy in the Task Table. The Energy Aggregator reduces the committed capacitor state to `ready`. The Dispatch Controller binds the result to a task and owns its execution window. The MCU participates only after these stages agree, leaving harvested energy available for application code rather than for the decision to start it.

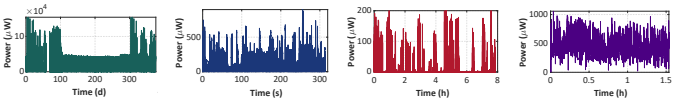
V. Evaluation

We evaluate HADES through four questions. **Q1** asks whether the mixed-signal admission path is accurate and conservative enough to control execution. **Q2** asks whether hardware-resident admission expands the operating envelope of intermittent systems. **Q3** isolates the mechanism-level sources of gain, including energy intake, bank-state reduction, and management overhead. **Q4** measures end-to-end value in throughput and completed tasks per joule. We then use ablations to connect the observed gains back to the major components of the design.

A. Methodology

We evaluate HADES using a unified mixed-signal SoC co-simulation flow that couples analog energy dynamics with cycle-accurate processor execution [52]. This closed-loop setup is essential because HADES's central claim is architectural rather than purely circuit-level: the physical energy state determines when execution is released, and processor activity in turn reshapes that state through capacitor discharge. Offline trace replay or purely software-level energy models [53], [54] break this feedback loop and therefore cannot faithfully capture the mechanism that HADES is designed to exploit. Our framework preserves it.

Analog modeling. We implement the Energy Intake Front-End, harvester-to-bank routing fabric, and Energy Aggregator in Verilog-A. The harvester is modeled as a voltage-dependent source interface whose operating point evolves with source characteristics, routing state, leakage, and digital load. At each timestep, capacitor-node voltages $\{V_{C,i}\}$ are updated according to harvested current, leakage, and instantaneous load current. We validate these voltage trajectories against analytical resistance-capacitance (RC) charge/discharge behavior under known load steps. The Energy Aggregator is designed in a commercial 180 nm PDK [55] and characterized with Cadence Spectre from Virtuoso ADE. Transistor-level DC and transient sweeps calibrate a compact Verilog-A model over the 1.5–3.3 V storage range, matching the evaluated openMSP430 supply range. The compact model is fitted at the quantities consumed by the architecture: the branch-sum transfer, the comparator boundary against V_{ref}, and the `ready` delay. The threshold table is generated over the supported bank-state domain so that each admitted code represents a conservative one-sided decision boundary rather than a global energy-ordering assumption. This calibration is performed at design time. DC sweeps over storage voltage, bank configuration, and V_{ref} fit the branch gain, offset, and comparator crossing, while transient voltage and load steps extract `ready` delay. The 204 DC points and 48 transients establish in-domain model fidelity; they are calibration cases rather than an independent generalization set. The flow completes in 3.6 hours on an Intel Xeon Silver 4216 16-core workstation and speeds workload-level AMS simulation by 11.4× compared with retaining the full transistor-level aggregator in the loop. The same characterization produces the nominal transfer used by $\mathcal{F}_{cal}$, and separate sweeps exercise branch mismatch, comparator



(a) Solar [57] (b) RF [58] (c) Kinetic [59] (d) Thermal [60]

Fig. 8. Real-world source traces used in evaluation.

offset, ladder shift, capacitance tolerance, and aging-equivalent drift around the decision boundary.

Digital modeling. The HADES Admission Substrate—including the Descriptor Ingress Path, Task Table, and Dispatch Controller—is implemented in synthesizable SystemVerilog as small always-on FSMs. We use an open-source openMSP430 core [56] as the managed compute substrate. In all experiments, the MSP430 executes the application together with any persistence and recovery code required by the intermittent runtime; voltage polling, admission, and admission-side scheduling are removed from software and handled entirely by hardware. We synthesize the digital blocks with Synopsys Design Compiler to obtain timing and area, and derive a post-synthesis power-state table using PrimePower in 180 nm at 8 MHz. During co-simulation, the testbench tracks processor state, including `Fetch`, `Decode`, `Execute`, and `Idle`, and updates the load current seen by the analog storage model accordingly. Accordingly, the processor is not modeled as a fixed load. Each simulated cycle contributes the current of its post-synthesis state to the capacitor-discharge model.

Mixed-signal integration. We instantiate the RTL and Verilog-A models in a single Cadence analog/mixed-signal (AMS) Designer testbench. The testbench emulates the edge tier by injecting E2D descriptors, including Task ID, Priority, and V_{ref}, through the Descriptor Ingress Path, which populates the Task Table. The Energy Aggregator produces the binary admission signal `ready`, which is fed directly to the Dispatch Controller. The Dispatch Controller then loads the selected threshold, generates `dispatch_grant`, drives the Atomicity Latch, and controls `grant`, `clk_en`, `reset_n`, and `ldo_en`. The LDO switch then gates the power path to the task load. All reported results come from this PDK-calibrated mixed-signal flow rather than fabricated silicon measurements. The flow couples RC storage dynamics, Spectre-fitted admission behavior, RTL control, and post-synthesis MSP430 power states in a single AMS testbench. The modeled hardware costs are reported in Sec. V-B, including admission-substrate logic, always-on FSMs, the Energy Aggregator interface, the configurable resistor ladder, and the LDO-control path. Behavioral calibration improves simulation fidelity and speed. It does not provide runtime oracle information to HADES.

Workloads, baselines, and metrics.

We drive the harvester models with the four open-source traces in Fig. 8, covering both quasi-continuous and burst-dominated regimes. Each trace is applied at the harvester terminals rather than injected as pre-stored capacitor energy. The current entering storage is determined by the source I–V behavior, the selected source–storage connection, capacitor voltage, leakage, and load current. Table I summarizes the 12 embedded workloads from MiBench and MediaBench. We evaluate them as bounded embedded-kernel task instances on the openMSP430 platform, rather than as full desktop-scale benchmark inputs, and check each admitted instance against its edge-generated isolation budget. We compare against InK [14], Failure Sentinels [6], Capybara [8], REACT [31], and CapDYN [9]. All systems use the same compute core, memory system, input traces, task binaries, and 47 μF total storage budget. For HADES and bank-aware baselines, this budget is organized as four equal nominal banks. Single-voltage baselines observe the same total storage through their native trigger or polling path. The 1.5–3.3 V operating range provides about 203 μJ of ideal stored energy before conversion loss and isolation margin. Reconfigurable-storage baselines run their published policies on this common fabric. We separately sweep storage size, bank heterogeneity, and bank count, including the $N = 4$–8 sweep in Fig. 19. We report three classes of metrics. For correctness, we measure compact-model fidelity at the comparator boundary and the false-positive rate of the final admission decision under variation and noise. A false positive is the critical event because it releases a task outside its profiled energy region; a false negative delays execution but preserves the conservative direction. For the operational envelope, we measure wake-to-first-instruction latency and minimum viable input power. For end-to-end value, we report throughput and energy efficiency in completed tasks per joule, defined as completed task instances per unit energy injected at the harvester terminals.

TABLE I
EMBEDDED BENCHMARK SUITE.

Category	Workloads	Primary stress points
General	Bitcnt, CRC32, Math, QSort	Short bursts, ALU wake-up, and recursive control
Network/Sec.	Dijk, Pat, AES, SHA	Irregular memory/control and long bounded-task computation
Signal/Media	FFT, Search, JPEG, GSM	Burst energy and mixed memory/compute

B. *Q1: Admission Correctness*

We first evaluate the admission path itself. A hardware-resident admission decision requires agreement between the transistor-level circuit and the compact model used in system simulation, a conservative comparator boundary under circuit and profile variation, and continuity of the execution grant after dispatch. The first two properties determine whether the task is released from a valid storage state. The third determines whether that decision remains effective after the task begins to draw current. The following experiments examine these properties separately and then report the area, power, and latency required to implement them.

Analog fidelity. We compare the compact Verilog-A model with the transistor-level Energy Aggregator over the characterized bank-state domain. For each state, the capacitor voltages define a reference usable-energy coordinate, and the circuit produces the aggregate output used by the comparator. Since

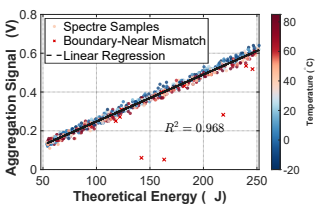


Fig. 9. Analog-model fidelity.

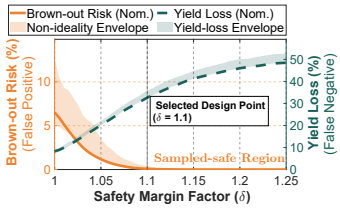


Fig. 10. Admission safety.

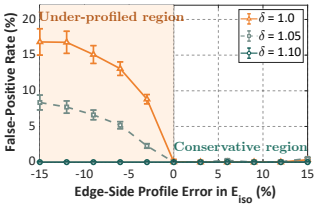


Fig. 11. Tolerance to edge-side profile error.

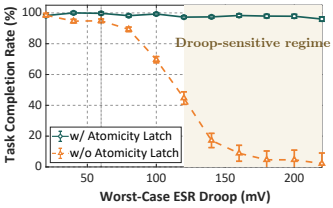


Fig. 12. Execution-window continuity.

the TIA is inverting, larger branch current lowers V_{sum}. Fig. 9 plots $\Delta V_{\mathrm{agg}} = V_{\mathrm{bias,d}} - V_{\mathrm{sum}}$, which increases with aggregate branch current.

Across the characterized voltage and temperature range, the compact model tracks the transistor-level transfer with $R^2 \approx 0.968$. This fit captures the monotonic movement of the aggregate signal as the capacitor state changes and supports closed-loop workload simulation without retaining the full transistor-level circuit in every run. The comparator boundary provides the direct admission metric. The compact and transistor-level models agree on 203 of the 204 characterized DC decisions. The single disagreement occurs 6.8 mV from the comparator boundary, where the conservative code-selection rule assigns the state to non-admission. Across 48 transient cases, the maximum difference in the `ready` transition is 0.41 μs. The Dispatch Controller uses a 1 μs settling interval before reset release, covering the measured transient discrepancy. These results establish the model accuracy used by the remaining mixed-signal experiments over the characterized operating domain.

Binary decision safety. A false positive releases the MCU from a state below the profiled task reserve; a false negative delays an otherwise feasible execution. We measure both quantities at the final binary output of the admission path. For each guardband $\delta \in [1.0, 1.25]$, the experiment evaluates 100,000 admission trials. The nominal sweep applies a bounded 5% perturbation to the calibrated aggregate output. The stressed sweep draws the residual circuit parameters from 180 nm Cadence Spectre characterization, including branch-gain error, comparator offset, transient noise, resistor-ladder reference shift, aging-equivalent gain drift, and capacitance tolerance. Fig. 10 reports the nominal result and the 5–95 percentile envelope across 80 stress realizations.

At $\delta = 1.1$, the false-positive rate is 0.09% in the nominal sweep and 0.38% at the 95th percentile of the stressed sweep. The corresponding false-negative rates are 32.5% and 35.4%. Increasing the guardband to $\delta = 1.13$ reduces the stressed false-positive rate below 0.1% in the evaluated variation model. The curves expose the expected monotonic tradeoff: a larger guardband suppresses unsafe release and postpones a larger fraction of otherwise feasible executions. Subsequent performance experiments use $\delta = 1.1$, the operating point marked in Fig. 10, and retain the measured safety and availability rates associated with that point. The selected operating point is applied consistently across all workloads and traces.

Hardware cost. Table II reports post-synthesis area, static power, dynamic decision energy, and response latency for the default 47 μF, four-bank configuration. The reported block includes the Energy Aggregator, intake-front-end comparison logic, configurable V_{ref} ladder, descriptor and Task Table support, Grant Logic, and the Atomicity Latch. The admission substrate occupies 3,455 μm^2, corresponding to 0.29% of the evaluated SoC area.

TABLE II
HARDWARE OVERHEAD ANALYSIS.

Module	Type	Comp. (GE/Trans.)	Area (μm^2)	Static (nW)	Dyn. (per dec.)	Lat. (resp.)
HADES *Subsystems*						
Ener. Aggr.	Mixed	58 Trans.	423	115	0.73 pJ	879 ns
Ener. Int. FE	Analog	12 Trans.	187	42	–	47 ns
Disp. Ctrl.	Digital	214 GE	2,845	385	12.40 pJ	1 cyc.
Total	**SoC blk.**	**214 GE + 70 Trans.**	**3,455**	**542**	**13.13 pJ**	**1.05 μs***
Software Baseline						
openMSP430	Logic	9,850 GE	131,005	4,210	–	–
Data Mem.	8KB SRAM	–	1,052,400	33,800	–	–
Total	**MCU**	**N/A**	**1,183,405**	**38,010**	**2.51 μJ**†	> 3.2 **ms**†

* Total response latency includes the Energy Aggregator, Energy Intake Front-End, and one 8 MHz Dispatch Controller cycle.
† Software admission includes bootloader execution and interrupt-service-routine polling for one admission attempt.

A comparison-and-grant event consumes 13.13 pJ of dynamic energy and completes in 1.05 μs. The always-on path draws 542 nW between decisions. For a waiting interval T_{wait}, the admission energy is therefore $13.13\,\mathrm{pJ} + 542\,\mathrm{nW} \times T_{\mathrm{wait}}$. The software baseline row reports the modeled cost of one MCU-side admission attempt, including bootloader execution and interrupt-driven polling. Its 2.51 μJ entry corresponds to active work performed during the attempt; the system-level evaluation additionally accounts for the time spent between attempts under each input trace. Separating static waiting power from event energy avoids assigning one fixed admission cost to traces with different charging intervals.

Profile robustness and margin sensitivity. The task threshold inherits uncertainty from the edge-generated isolation budget. We quantify this sensitivity by perturbing E_{iso} with a relative profile error $\epsilon \in [-15\%, +15\%]$ before mapping it to V_{ref}. Negative values represent under-profiling and produce a more permissive threshold; positive values increase the task reserve. Each setting contains 1,000 mixed-signal trials with randomized PVT variation, analog noise, and bursty input traces. The reported metric is the false-positive rate at the final admission output.

Fig. 11 shows that under-profiling increases unsafe ad-

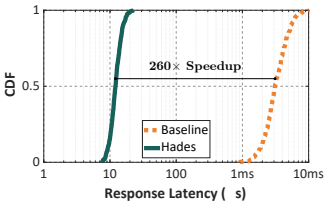


Fig. 13. Burst-to-execution latency.

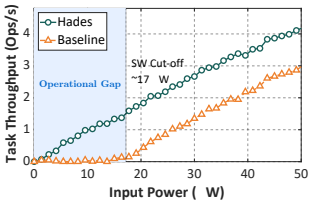


Fig. 14. Minimum viable input power.

missions when $\delta = 1.0$. The effect follows directly from the threshold mapping: an underestimated task budget moves V_{ref} toward a more permissive code and admits lower-energy states. The added isolation margin shifts this boundary toward conservative decisions. At $\delta = 1.1$, no false positive is observed in the 1,000 profile-error trials. Positive profile error lowers utilization because the task waits for additional stored energy, while the admission direction remains safe. The larger experiment in Fig. 10 supplies the corresponding circuit-variation statistics with 100,000 trials per guardband. Together, the two sweeps separate profile uncertainty from residual analog variation and show how δ controls their combined effect on admission.

Execution-window continuity. Admission occurs before the task load is enabled. During execution, capacitor voltage falls as the task consumes the reserved energy, and V_{sum} can cross the programmed threshold in the reverse direction. A grant that continuously follows `ready` would then terminate an execution window that had already passed admission.

We compare the full design with a latch-free variant whose grant tracks `ready` throughout execution. The two variants use the same task certificates, bank states, and power path; the grant-retention policy is the only changed component. The experiment sweeps the equivalent ESR-induced voltage droop at the task load and replays randomized burst traces under the same task profiles. Fig. 12 reports the completed-task fraction. The latch-free variant loses completion rate as droop increases because the comparator deasserts the grant after dispatch. With the Atomicity Latch, the execution window remains active until `task_done`, and completion stays near 100% over the characterized range. The task certificate and supply envelope establish the energy condition at dispatch; the latch preserves that admitted window while the reserve is consumed.

C. *Q2: Operational Envelope*

We next measure how the admission path changes the range of energy opportunities that reach application execution. The experiments cover response to short bursts and the minimum input power that sustains task completion.

Responsiveness to short energy bursts. We measure the interval from the first point at which harvested input exceeds platform leakage, $P_{\text{in}} > P_{\text{leak}}$, to the first application instruction fetch. This end-to-end latency includes the architecture-specific charge accumulation, admission, power sequencing, and software startup that occur after the burst begins. It therefore captures the full response visible to the application under each design. Fig. 13 reports the CDF over 1,000 randomized wakeup events. InK reaches a median of approximately 3.2 ms, dominated by oscillator stabilization, boot, and runtime initialization. Its upper tail reflects events in which the accumulated burst is partly consumed before admission completes. HADES reaches the first instruction in a median of 12 μs through direct hardware admission and sequenced reset release. The shorter path allows brief input bursts to reach application execution before the available energy is consumed by startup activity.

Minimum viable input power. We sweep input power from 1 μW to 50 μW and measure task completion over the fixed evaluation interval used for all systems. The minimum viable point is the lowest input level that maintains a non-zero completion rate for the profiled task stream. Each system retains its own admission mechanism while using the same task image and storage budget. Fig. 14 places the software-managed boundary near 17 μW. Below this point, wakeup and polling consume the energy accumulated between execution attempts, and the task stream reaches a terminal stall. HADES continues to complete bounded tasks at approximately 2 μW. At lower power, the 542 nW always-on path, storage leakage, and the remaining platform leakage consume the available input before the next task reserve is reached.

D. *Q3: Mechanism Effectiveness*

The end-to-end result combines energy intake, storage-state interpretation, and admission overhead. We isolate these mechanisms before reporting application throughput.

Income-side gains from the Energy Intake Front-End. We evaluate source–storage matching with a $V_{OC} = 3.0$ V solar source and a $V_{OC} = 1.0$ V RF source. The voltage separation creates repeated intervals in which one static source–bank assignment favors one source and penalizes the other. The fixed-pairing baseline retains that assignment throughout the trace. HADES changes the assignment when the characterized voltage windows identify a more favorable pair. Fig. 15 integrates energy delivered to storage over 4 s after switch R_{on} loss. Fixed pairing repeatedly clamps a source to a mismatched capacitor voltage, while the matching front end moves each source toward its productive region. The cumulative delivered energy increases by 1.9× under the evaluated trace. Since the metric integrates delivered energy, repeated charging and task discharge during the interval are represented without exceeding the instantaneous capacity of the bank.

Admission-side gains from analog bank-state reduction. Fig. 16 compares the Energy Aggregator with local voltage-threshold admission under the same distributed capacitor state. A local threshold waits for one observed node to cross its trigger and therefore depends on how the total charge is partitioned. The aggregate circuit combines the bank voltages in parallel and evaluates the calibrated task boundary from the complete bank state. When charge is spread across several banks, the aggregate signal reaches the task threshold before any individual node reaches the local trigger. The waveform

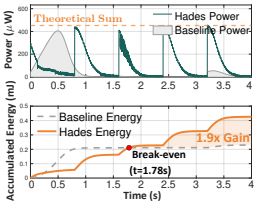


Fig. 15. Source–storage matching gain.

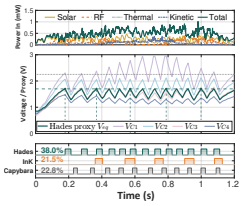


Fig. 16. Distributed-state admission.

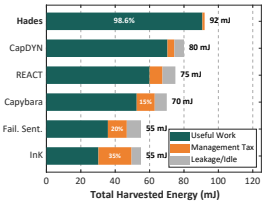


Fig. 17. Energy consumption breakdown.

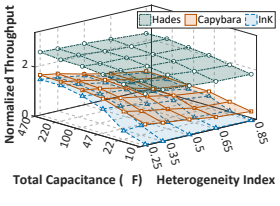


Fig. 18. Storage-organization effects.

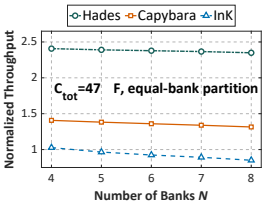


Fig. 19. Bank-count sensitivity.

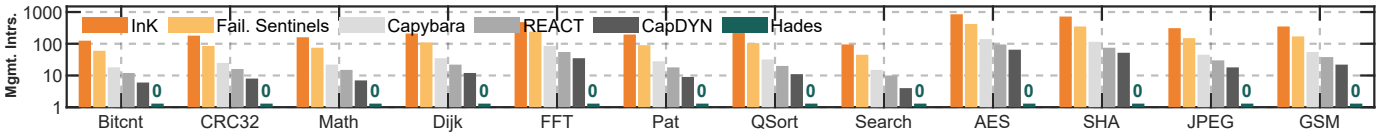


Fig. 20. Admission-management interrupts per completed task.

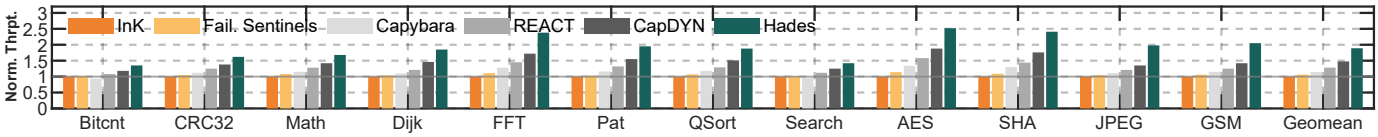


Fig. 21. System task throughput.

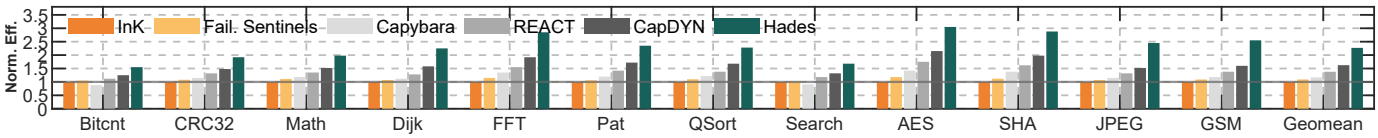


Fig. 22. Completed tasks per joule at the harvester terminals.

shows the resulting advance in `ready`, reset release, and task execution.

Management-side energy breakdown. Fig. 17 partitions total energy into application execution, admission and runtime management, and the remaining platform components. The accounting uses the same harvester-terminal input boundary as the tasks/J metric. Software baselines spend a visible portion on MCU wakeup, polling, and scheduling activity. In HADES, the 542 nW always-on path replaces repeated MCU activation during the waiting period, and the 13.13 pJ event cost is incurred only when the controller evaluates and releases a task. The resulting distribution assigns a larger fraction of the harvested budget to application execution.

Sensitivity to storage organization. The default storage budget is 47 μF divided across four equal banks. Fig. 18 varies total capacitance from 10 μF to 470 μF and changes the four-bank partition using

$$H = \frac{\max_i C_i}{\sum_i C_i}.$$

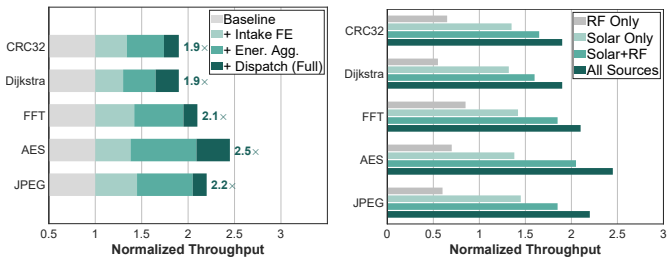
Here, $H = 0.25$ denotes a uniform four-bank layout, and larger values place a greater fraction of the total capacitance in the dominant bank. Each point uses the same concrete partition across all compared systems. Software-managed designs lose the largest fraction of throughput at small C_{tot}, where each wakeup consumes more of the available burst. Local-threshold designs also become sensitive to increasingly uneven bank voltages as H grows.

Fig. 19 fixes $C_{\text{tot}} = 47\,\mu$F and sweeps equal-bank organizations from $N = 4$ to $N = 8$. Each increase in N reduces the nominal capacitance of an individual bank and increases the fan-in presented to the analog summation node. HADES shows modest throughput variation over this range. The associated settling and mismatch effects remain within the calibrated envelope evaluated in Fig. 10. Baselines that reconstruct or trigger on local bank state incur more polling, threshold delay, or reconfiguration work as the same storage budget is divided across additional banks. The two sweeps establish the evaluated range over which the aggregate interface preserves its system-level benefit.

E. Q4: End-to-End System Value

We now report the application-level effects of the complete design: admission-management activity, task throughput, and completed tasks per joule.

Admission-management interrupts. Fig. 20 counts MCU interrupts generated by the admission and runtime-management path for each completed task. This normalization exposes the management work required to obtain one successful application completion under each trace. InK produces hundreds of timer-driven interrupts on long kernels such as AES and FFT. Hardware-assisted baselines reduce polling frequency,



(a) Component ablation (b) HADES source mix

Fig. 23. Ablation study.

while readiness events and task-selection updates still enter the MCU control path. HADES performs admission without MCU polling and records zero admission-management interrupts across the 12 workloads. Application-defined sensor, timer, and radio interrupts remain part of task execution and are excluded from this count.

Application throughput. Fig. 21 reports completed task instances per unit time, normalized to InK. The normalization is performed per workload before computing the geometric mean. HADES achieves 2.2× the geometric-mean throughput of InK and 1.35× that of CapDYN. AES and SHA reach up to 2.5× InK because their longer execution windows avoid repeated management entry. Dijkstra and the other control- or memory-intensive workloads show gains around 1.5×, reflecting lower wakeup and scheduling overhead across irregular execution phases. The positive result across all 12 workloads indicates that the gain is not confined to one execution pattern.

End-to-end energy efficiency. Fig. 22 reports completed tasks per joule injected at the harvester terminals. This boundary includes source–storage matching, storage loss, admission activity, and application execution, and therefore measures the efficiency of the complete energy path. HADES improves the geometric-mean efficiency by 2.4× over InK. The gain combines the additional energy delivered by the intake front end with the lower energy spent on MCU-side admission. Workloads with long waiting intervals benefit from the lower recurring management power, while burst-heavy workloads also benefit from the shorter release path. The breakdown in Fig. 17 and the component ablation below separate these contributions.

F. Ablation Study

The ablation study removes or adds one mechanism at a time under the same workload, storage, and trace configuration.

Architectural ablation. The starting configuration uses fixed source pairing, local voltage-threshold monitoring, round-robin task selection, and software admission. Every subsequent point retains the preceding mechanisms and adds one architectural component. Adding the Energy Intake Front-End increases average throughput to 1.38× the starting point by reducing source–storage mismatch. Adding the Energy Aggregator raises the result to 1.9× through earlier admission from distributed bank states. The complete design adds the Dispatch Controller and reaches up to 2.45× on the workloads with the largest admission-management component. The monotonic increments match the mechanism experiments: source matching increases stored energy, aggregate reduction exposes usable distributed charge, and hardware dispatch removes MCU-side admission work.

Source-mix adaptability. Fig. 23(b) evaluates RF-only, solar-only, solar+RF, and all-source inputs, with throughput normalized to the solar-only software-admission baseline. RF-only operation produces the lowest throughput because its bursts are weaker and shorter in the evaluated traces. Solar+RF and all-source configurations increase both the available input energy and the opportunities for source–storage matching. The throughput ordering follows these two effects: richer source mixes raise the energy delivered to storage and reduce intervals in which one fixed load point constrains all sources. Across the evaluated mixes, the same comparator and routing path adapts the charging topology without source-specific MCU control.

VI. LIMITATIONS

First, HADES assumes that each task can be assigned a conservative admission threshold through offline or edge-side profiling. This works well for many embedded workloads with repetitive and bounded behavior, but is less precise for highly data-dependent tasks whose energy varies substantially across inputs. In such cases, guaranteeing atomicity requires padding toward worst-case execution, which may leave some harvested energy unused. HADES therefore trades admission precision for lower runtime management overhead. Second, our current implementation is realized in a 180 nm CMOS process. This is not merely a prototyping choice, but reflects the low-leakage and more predictable analog behavior that the current Energy Aggregator relies on. The architectural idea of hardware-resident admission is not inherently tied to 180 nm, but the present aggregator circuit is. In deeply scaled FinFET nodes, short-channel effects and velocity saturation would significantly alter the voltage-to-current relationship used in our design. Porting HADES to such technologies would therefore require redesigning the aggregation circuit.

VII. CONCLUSION

This paper presents HADES, an autonomous admission substrate that moves energy admission out of the MCU software path and into an always-on mixed-signal control plane. In doing so, HADES redefines admission as a hardware function, reducing control overhead and improving how intermittently powered systems translate harvested energy into useful work.

ACKNOWLEDGMENT

We sincerely thank all anonymous reviewers for their time and efforts in reviewing our work and providing valuable feedback. This work was supported by the National Natural Science Foundation of China under Grant 62572098.

REFERENCES

[1] B. Denby and B. Lucia, "Orbital edge computing: Nanosatellite constellations as a new class of computer system," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 939–954.

[2] J. Jang and F. Adib, "Underwater backscatter networking," in *Proceedings of the ACM special interest group on data communication*, 2019, pp. 187–199.

[3] S. S. Afzal, W. Akbar, O. Rodriguez, M. Doumet, U. Ha, R. Ghaffarivardavagh, and F. Adib, "Battery-free wireless imaging of underwater environments," *Nature communications*, vol. 13, no. 1, p. 5546, 2022.

[4] S. Ahmed, B. Islam, K. S. Yildirim, M. Zimmerling, P. Pawełczak, M. H. Alizai, B. Lucia, L. Mottola, J. Sorber, and J. Hester, "The internet of batteryless things," *Communications of the ACM*, vol. 67, no. 3, pp. 64–73, 2024.

[5] G. Gobieski, B. Lucia, and N. Beckmann, "Intelligence beyond the edge: Inference on intermittent embedded systems," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 199–213.

[6] H. Williams, M. Moukarzel, and M. Hicks, "Failure sentinels: Ubiquitous just-in-time intermittent computation via low-cost hardware support for voltage monitoring," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 665–678.

[7] S. Li, L. Lu, M. J. Hussain, Y. Ye, and H. Zhu, "Sentinel: Breaking the bottleneck of energy utilization efficiency in rf-powered devices," *IEEE Internet of Things Journal*, vol. 6, no. 1, pp. 705–717, 2018.

[8] A. Colin, E. Ruppel, and B. Lucia, "A reconfigurable energy storage architecture for energy-harvesting devices," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, 2018, pp. 767–781.

[9] M. Doglioni, E. Yildiz, M. Nardello, K. Akhunov, K. S. Yildirim, and D. Brunelli, "Capdyn: Adaptive self-scaling energy storage for powering batteryless iot," *ACM Transactions on Embedded Computing Systems*, vol. 24, no. 5, pp. 1–32, 2025.

[10] J. de Winkel, C. Delle Donne, K. S. Yildirim, P. Pawełczak, and J. Hester, "Reliable timekeeping for intermittent computing," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 53–67.

[11] C. Park and P. H. Chou, "Ambimax: Autonomous energy harvesting platform for multi-supply wireless sensor nodes," in *2006 3rd annual IEEE communications society on sensor and ad hoc communications and networks*, vol. 1. IEEE, 2006, pp. 168–177.

[12] P. P. Puluckul and M. Weyn, "Infiniteen: A multi-source energy harvesting system with load monitoring module for batteryless internet of things," in *2023 IEEE 9th World Forum on Internet of Things (WF-IoT)*. IEEE, 2023, pp. 1–8.

[13] M. Buettner, B. Greenstein, and D. Wetherall, "Dewdrop: An {Energy-Aware} runtime for computational {RFID}," in *8th USENIX Symposium on Networked Systems Design and Implementation (NSDI 11)*, 2011.

[14] K. S. Yıldırım, A. Y. Majid, D. Patoukas, K. Schaper, P. Pawelczak, and J. Hester, "Ink: Reactive kernel for tiny batteryless sensors," in *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*, 2018, pp. 41–53.

[15] K. Maeng and B. Lucia, "Adaptive low-overhead scheduling for periodic and reactive intermittent execution," in *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2020, pp. 1005–1021.

[16] B. Ransford, J. Sorber, and K. Fu, "Mementos: System support for long-running computation on rfid-scale devices," in *Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems*, 2011, pp. 159–170.

[17] K. Maeng, A. Colin, and B. Lucia, "Alpaca: Intermittent execution without checkpoints," *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, vol. 1, pp. 1–30, 2017.

[18] A. Colin and B. Lucia, "Chain: tasks and channels for reliable intermittent programs," in *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, 2016, pp. 514–530.

[19] J. Van Der Woude and M. Hicks, "Intermittent computation without hardware support or programmer intervention," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 17–32.

[20] K. Maeng and B. Lucia, "Adaptive dynamic checkpointing for safe efficient intermittent computing," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 2018, pp. 129–144.

[21] A. Bakar, A. G. Ross, K. S. Yildirim, and J. Hester, "Rehash: A flexible, developer focused, heuristic adaptation platform for intermittently powered computing," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 5, no. 3, pp. 1–42, 2021.

[22] V. Kortbeek, K. S. Yildirim, A. Bakar, J. Sorber, J. Hester, and P. Pawełczak, "Time-sensitive intermittent computing meets legacy software," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 85–99.

[23] K. Maeng and B. Lucia, "Supporting peripherals in intermittent systems with just-in-time checkpoints," in *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2019, pp. 1101–1116.

[24] E. Yıldız, L. Chen, and K. S. Yıldırım, "Immortal threads: Multithreaded event-driven intermittent computing on {Ultra-Low-Power} microcontrollers," in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, 2022, pp. 339–355.

[25] Y. Wu, B. Min, M. Ismail, W. Xiong, C. Jung, and D. Lee, "{IntOS}: Persistent embedded operating system and language support for multithreaded intermittent computing," in *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, 2024, pp. 425–443.

[26] E. Yildiz, K. Akhunov, L. A. Riva, A. Goknil, I. Kurtev, and K. S. Yildirim, "Adaptable runtime monitoring for intermittent systems," in *Proceedings of the Nineteenth European Conference on Computer Systems*, 2024, pp. 1175–1191.

[27] H. Desai, X. Wang, and B. Lucia, "Energy-aware scheduling and input buffer overflow prevention for energy-harvesting systems," in *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, 2025, pp. 339–354.

[28] E. Ruppel, M. Surbatovich, H. Desai, K. Maeng, and B. Lucia, "An architectural charge management interface for energy-harvesting systems," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 318–335.

[29] A. Alsubhi, S. Babatunde, N. Tobias, and J. Sorber, "Stash: Flexible energy storage for intermittent sensors," *ACM Transactions on Embedded Computing Systems*, vol. 23, no. 2, pp. 1–23, 2024.

[30] F. Yang, A. S. Thangarajan, S. Michiels, W. Joosen, and D. Hughes, "Morphy: Software defined charge storage for the iot," in *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*, 2021, pp. 248–260.

[31] H. Williams and M. Hicks, "Energy-adaptive buffering for efficient, responsive, and persistent batteryless systems," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, 2024, pp. 268–282.

[32] M. Hicks, "Clank: Architectural support for intermittent computation," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, 2017, pp. 228–240.

[33] J. Zeng, J. Choi, X. Fu, A. P. Shreepathi, D. Lee, C. Min, and C. Jung, "Replaycache: Enabling volatile cachesfor energy harvesting systems," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 170–182.

[34] A. Bhattacharyya, A. Somashekhar, and J. S. Miguel, "Nvmr: nonvolatile memory renaming for intermittent computing," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 1–13.

[35] J. Choi, J. Zeng, D. Lee, C. Min, and C. Jung, "Write-light cache for energy harvesting systems," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–13.

[36] Y. Zhou, J. Zeng, J. Jeong, J. Choi, and C. Jung, "Sweepcache: Intermittence-aware cache on the cheap," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, 2023, pp. 1059–1074.

[37] Y. Zhou, J. Zeng, and C. Jung, "Lightwsp: Whole-system persistence on the cheap," in *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2024, pp. 215–230.

[38] N. Hassan, B. Min, C. Jung, Y. Solihin, and J. Choi, "Warmcache: Exploiting stt-ram cache for low-power intermittent systems," in *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, 2025, pp. 586–600.

[39] S. Mohapatra, V. Kortbeek, M. A. van Eerden, J. Broekhoff, S. Ahmed, and P. Pawełczak, "Data cache for intermittent computing systems with non-volatile main memory," in *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, 2025, pp. 227–243.

[40] G. Fang, J. Zeng, A. Gupta, and C. Jung, "Rethinking prefetching for intermittent computing," in *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, 2025, pp. 225–240.

[41] G. Fang and C. Jung, "Rethinking dead block prediction for intermittent computing," in *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2025, pp. 732–744.

[42] J. Choi, Q. Liu, and C. Jung, "Cospec: Compiler directed speculative intermittent computation," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 399–412.

[43] J. Choi, L. Kittinger, Q. Liu, and C. Jung, "Compiler-directed high-performance intermittent computation with power failure immunity," in *2022 IEEE 28th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2022, pp. 40–54.

[44] G. Fang, J. Choi, and C. Jung, "Hybrid power failure recovery for intermittent computing," in *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, 2024, pp. 1–9.

[45] K. Akhunov and K. S. Yildirim, "Adamica: Adaptive multicore intermittent computing," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 6, no. 3, pp. 1–30, 2022.

[46] K. Akhunov, E. Yildiz, and K. S. Yildirim, "Pearl: Power-and energy-aware multicore intermittent computing," in *The International Conference on Embedded Wireless Systems and Networks (EWSN)*, 2025, pp. 1–1.

[47] S. Bandyopadhyay and A. P. Chandrakasan, "Platform architecture for solar, thermal, and vibration energy combining with mppt and single inductor," *IEEE journal of solid-state circuits*, vol. 47, no. 9, pp. 2199–2215, 2012.

[48] A. Devaraj, M. Megahed, Y. Liu, A. Ramachandran, and T. Anand, "A switched capacitor multiple input single output energy harvester (solar+ piezo) achieving 74.6% efficiency with simultaneous mppt," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 66, no. 12, pp. 4876–4887, 2019.

[49] D. Khan, S.-J. Oh, S. Yeo, Y. Ryu, S.-H. In, R. E. Rad, I. Ali, Y.-G. Pu, S.-S. Yoo, M. Lee, K. C. Hwang, Y. Yang, and K.-Y. Lee, "A high-efficient wireless power receiver for hybrid energy-harvesting sources," *IEEE Transactions on Power Electronics*, vol. 36, no. 10, pp. 11 148–11 162, 2021.

[50] L. Mamouri, T. Mesbahi, and V. Frick, "Mppt technique with improved focv control applied to a full active rectifier for low-voltage piezoelectric energy harvesting," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 71, no. 2, pp. 532–536, 2023.

[51] N. Femia, G. Petrone, G. Spagnuolo, and M. Vitelli, "Optimization of perturb and observe maximum power point tracking method," *IEEE transactions on power electronics*, vol. 20, no. 4, pp. 963–973, 2005.

[52] R. Gretsch, M. Beyeler, J. Lau, and T. Sherwood, "Single spike artificial neural networks," in *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, 2025, pp. 916–929.

[53] ink anonymous, "Ink source code," https://github.com/TUDSSL/InK, 2017.

[54] FoRTE-Research, "React source code," https://github.com/FoRTE-Research/REACT-Artifact, 2024.

[55] TSMC, "180nm cmos technology," https://www.tsmc.com/english/dedicatedFoundry/technology/logic/l_018micron, 2026.

[56] O. Girard, "openmsp430," https://github.com/olgirard/openmsp430, 2021.

[57] EnHANTs, "Solar power traces," https://enhants.ee.columbia.edu/indoor-irradiance-meas, 2011.

[58] FoRTE-Research, "Rf power traces," https://github.com/FoRTE-Research/REACT-Artifact/tree/master/traces, 2024.

[59] EnHANTs, "Kinetic power traces," https://enhants.ee.columbia.edu/kinetic-datasets, 2014.

[60] V. A. Leal Sobral, J. Lach, J. L. Goodall, and B. Campbell, "Thermal power traces," https://doi.org/10.18130/V3/M9CP9C, 2021.