Poster: Automating Approximations in Batteryless IoT Devices

Abdullah Soomro
Department of Computer Science
LUMS
Pakistan

Naveed Anwar Bhatti
Muhammad Hamad Alizai
Department of Computer Science
LUMS
Pakistan

Abstract

Batteryless IoT devices powered by energy harvesting face operational challenges due to intermittent power. Traditional checkpointing, while essential for state preservation, incurs high energy and time costs. We designed Approxify, an automated framework, that reduces energy consumption by 40% using intelligent approximation techniques. It optimizes energy-accuracy trade-offs, ensuring reliable operation without compromising key functions. Evaluations on three applications show significant reductions in checkpoint frequency and energy usage while maintaining acceptable error margins.

CCS Concepts

• **Computer systems organization** → **Embedded software**; • **Computing methodologies** → **Information extraction**; • **Theory of computation** → **Approximation algorithms analysis**.

Keywords

Transiently-Powered Embedded Devices (TPES), Approximate Computing, Large Language Models (LLMs)

1 Introduction and Design Goals

Batteryless IoT devices, powered by energy harvesting, introduce significant challenges for computation and communication, highlighting the importance of energy efficiency and continuous operation [4]. Traditional checkpointing methods, while useful for coping with power interruptions by saving and restoring system states, impose considerable overheads. Approximate computing complements checkpointing by balancing computational accuracy and energy efficiencies [2].

Nonetheless, applying approximation techniques in batteryless IoT systems is challenging due to several factors. First, complex manual tuning is required for each application, making the process time-consuming and prone to inaccuracies. Second, identifying suitable approximation opportunities demands a deep understanding of both the application and techniques to balance energy efficiency and reliability. Finally, assessing the impact and predicting program behavior post-deployment is difficult, complicating the scalability and adaptability of solutions.

We present *Approxify*, a tool that automates approximation techniques to reduce checkpoint frequency. It is designed with the following goals to apply approximate computing in batteryless embedded systems effectively:① Generalizes approximation opportunities across various applications, ② Simplifies the developer's experience by automating approximation strategies, ③ Balances energy efficiency and accuracy with a variety of techniques, and ④ Adjusts approximation levels during compile time using realistic energy traces

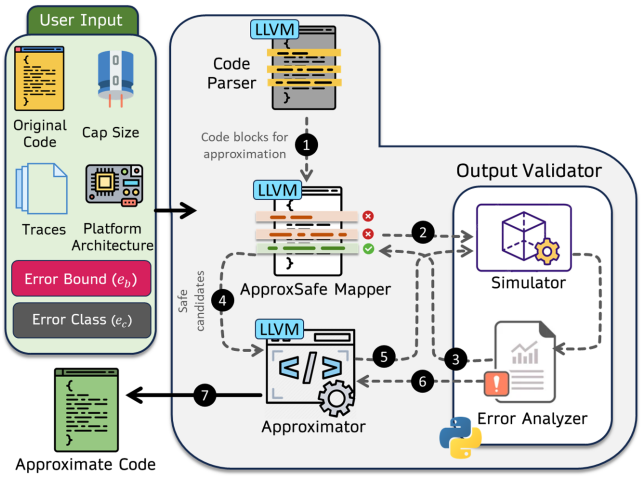


Figure 1: *Approxify* architecture.

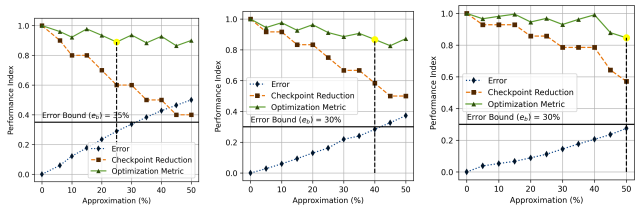
2 *Approxify*: System Architecture

Approxify takes six key inputs from the user: *original application source code* for identifying approximation opportunities, *input trace* to simulate application inputs, *error class* (e_c) and *error bound* (e_b) for managing the trade-off between accuracy and energy efficiency, *platform architecture* (e.g., ARM Cortex M or MSP430) for energy consumption estimation, and *capacitor size* to calculate the minimum energy required to avoid non-progressive states. These inputs enable Approxify to balance accuracy and energy efficiency in batteryless IoT devices, as shown in Figure 1. Approxify's core components work together to automate and optimize approximations:

Code Parser: The Code Parser identifies key code blocks—such as variables, functions, and loops—required for approximation. It systematically applies approximation techniques, ensuring all potential areas for optimization are considered.

Output Validator: The Output Validator evaluates the performance of approximations using a simulator and error analyzer. The simulator determines checkpoint frequency and energy usage, while the error analyzer measures the accuracy deviation, ensuring the selected approximations meet energy and accuracy requirements.

ApproxSafe Mapper: The ApproxSafe Mapper identifies code blocks safe for approximation by evaluating their impact on performance and accuracy. It filters out blocks that don't improve



(a) SUSAN - 220μF **(b) LQI** - 220μF **(c) String Search** - 47μF

Figure 2: Performance index vs. approximation percentage for three applications with smallest capacitors.

efficiency or exceed the error threshold, focusing only on blocks that enhance overall system performance.

Approximator: The Approximator fine-tunes safe approximations, adjusting intensity levels iteratively to balance energy savings and accuracy. It ensures that approximation-induced errors stay within acceptable limits while maximizing checkpoint reductions.

3 Implementation

Approxify leverages LLVM [3] for code parsing and compiler passes, paired with a Python framework for simulating applications. This combination allows efficient source code modifications and accurate approximation evaluations. LLVM's abstract syntax tree (AST) is used to identify essential code blocks, isolating those suitable for approximation. Compiler passes modify the source code, and only blocks that improve the checkpoint reduction ratio (c) or meet the error threshold (e_b) are selected for further processing. Using `Renode` [6] and `MSPSim` [1], Approxify simulates approximated code across various architectures, ensuring generalizability for batteryless IoT platforms. The checkpointing simulator, based on `MementOS` [5], mimics energy consumption using user-supplied voltage traces and capacitor sizes, predicting necessary checkpoints. Approxify 's optimization is agnostic to the checkpointing system, aiming to reduce computational cycles and enhance energy efficiency across any system.

4 Preliminary Results & Next Steps

Simulation Results: We evaluated Approxify's performance across three applications—SUSAN, Link Quality Indicator (LQI), and String Search—on four different capacitor sizes as input to the system. However, due to space constraints, we report the results for the smallest capacitors. The evaluations were conducted using the STM32 L152RE board with a maximum error bound of 35%.

For SUSAN, a widely used algorithm in IoT for edge detection, Approxify reduced checkpoint frequency by 40% with a 220μF capacitor while maintaining a 28% error margin using the Structural Similarity Index (SSIM). Similarly, the LQI, crucial for wireless communication reliability, showed a 33% checkpoint reduction with a 220μF capacitor, maintaining an error within 28%. Lastly, the String Search, evaluated with a 47μF capacitor, demonstrated a 20% reduction in checkpoints while keeping the error within acceptable limits based on the F1 score. These results highlight Approxify's ability to optimize energy efficiency while controlling accuracy across different IoT applications.

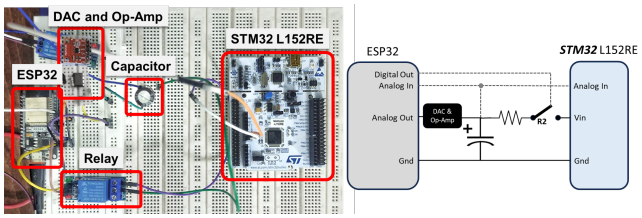


Figure 3: Testbed / Schematic for checkpointing validation.

Table 1: Number of checkpoints on multiple energy traces for SUSAN, LQI, and String Search applications. Each cell contains three values: the number of checkpoints on the testbed (black), predicted by Approxify's simulator (cyan), and the original without approximations (red).

Traces	SUSAN		LQI		String Search	
	220µF	330µF	220µF	330µF	47µF	68µF
RF#1	6/6/10	2/2/3	7/7/10	3/3/4	8/8/10	2/2/4
RF#2	6/6/10	2/2/3	7/7/10	3/3/4	8/8/10	2/2/4
RF#3	6/6/10	2/2/3	7/7/10	3/3/4	9/9/11	2/2/4
RF#4	6/6/10	2/2/3	7/7/10	3/3/4	8/8/10	2/2/4
RF#5	6/6/10	2/2/3	7/7/10	3/3/4	8/8/10	2/2/4

Testbed Validation: To verify Approxify's performance, we deployed a testbed that replicates real-world energy harvesting scenarios, ensuring accuracy and repeatability in diverse energy environments. The testbed consists of an ESP32 microcontroller, which controls the energy supply to STM32 L152RE board as shown in Figure 3. Energy flow is managed by a relay controlled by the ESP32, which connects the STM32 board to a capacitor, closely simulating real-world capacitor charge and discharge cycles.

We used five distinct RF energy traces from MementOS [5]. These traces were replayed on the testbed, with the ESP32 monitoring capacitor voltage at 1 ms intervals to manage the relay and charge-discharge phases, while the STM32 board dynamically triggered checkpoints via `MementOS`. The summary of the results of five energy traces is available in Table 1.

What's Next? We are exploring the use of Large Language Models (LLMs) to achieve fine-tuning that approaches human-level precision in approximation. In the future, Approxify will include adaptive decision policies for real-time energy trends.

References

[1] J. Eriksson and et al. Poster abstract: Mspsim–an extensible simulator for msp430-equipped sensor boards. 01 2007.
[2] K. Javed and et al. Moptic-sm: Sleep mode-enabled multi-optimized intermittent computing for transiently powered systems. *Journal of Systems Architecture*, 137:102850, 2023.
[3] C. Lattner and V. Adve. LLVM: A compilation framework for lifelong program analysis & transformation. In *CGO*, 2004.
[4] B. Lucia and et al. Intermittent computing: Challenges and opportunities. *SNAPL 2017*, 2017.
[5] B. Ransford and et al. Mementos: system support for long-running computation on rfid-scale devices. In *ASPLOS*, page 159–170, 2011.
[6] Renode. Simulation framework for complex embedded systems.