UDC 004.032.2 **doi: 10.32620/reks.2026.2.03**

Oleksandr MAMCHYCH, Maksym VOLK

Kharkiv National University of Radioelectronics, Kharkiv, Ukraine

A TOPOLOGY-AWARE UNIFIED MODEL AND METHOD FOR FORECASTING ENERGY CONSUMPTION IN DISTRIBUTED SYSTEMS WITH STATIONARY AND MOBILE DEVICES

*The **subject of this research** is energy consumption modeling and forecasting in distributed computing systems using Entity-Component Architecture (ECA). **This task** focuses on improving the **method** of quantifying and predicting the energy demands of a distributed computing system by simulating all components of this system, such as computational units (CPUs, GPUs), data transmission interfaces (LAN, USB interfaces), and network infrastructure (LAN switches, routers) in heterogeneous environments. Method **validation** is based on the stationary computing network (SCN) and mobile computing network (MCN) cases. This study addresses the inherent limitations of traditional algebraic models by advocating for a modular, scalable, and flexible ECA-based framework that integrates direct and indirect power expenditures of distributed computational elements to facilitate energy profiling. **The experimental validation** showed that the proposed ECA model achieves high prediction accuracy. In SCN configurations, simulations matched real-world energy measurements within 4.4% for CPU-only tasks, while CPU+GPU tasks showed a larger discrepancy of 20.2% due to GPU underutilization. Conversely, the MCN scenario, involving mobile devices and USB connections, resulted in minimal deviations of 3.5% in CPU-only mode and approximately 10.1% in CPU+GPU mode. These results underscore the model's robustness in accurately forecasting energy usage across diverse computational settings. We **conclude** that the effective transition from conventional algebraic models to a flexible and extendable ECA approach significantly enhanced the methodology's modularity, scalability, and universality. Major achievements include refining computational benchmarking practices by generalizing computing payloads, adopting all-inclusive energy measurements, and highlighting the energy-efficiency advantage of computations on mobile platforms over traditional stationary hardware for both CPU and GPU tasks. This research has highlighted some **limitations** of current methods, such as linear interpolation in load-energy modeling and the neglect of momentary hardware states, including momentary temperature and temperature inertia, which affect performance and consumption. The **scientific novelty** of the research consists of two parts: a new approach to mathematical modeling that enables extension and scalability, and **a more consistent and accurate model** of a distributed computing system based on it.*

Keywords: *energy efficiency; distributed computing; cloud computing; green computing; mathematical modeling; smartphone; central processing unit; graphical processing unit; entity-component architecture.*

1. Introduction

The 2024 United States Data Center Energy Usage Report by Lawrence Berkeley National Laboratory [1] reveals a significant rise in data center electricity consumption, accounting for 4.4% of the nation's total electricity use. Projections indicate that this could increase to 325-580 TWh, representing up to 12% of U.S. electricity consumption by 2028. Network infrastructure within data centers is projected to consume 23 TWh by 2028, driven by AI-related data traffic. This underscores the growing importance of energy-efficient networking solutions to mitigate the increasing power demands of expanding data center operations.

The measurement of energy consumption in distributed computing networks is a complex task. It should include not only the consumption of direct computing units but also the consumption of all components: networks, cooling systems, and power supply units. It is difficult to predict every component that we will need to account for. This is why we set the main goals in this research: to make the existing method of energy consumption forecasting in distributed computing systems more flexible and extendable.

1.1. Motivation

Many domains, including scientific research, show steady growth in the demand for computing resources [2]. High-performance computing not only increases in capabilities but also in energy consumption. This issue becomes even more critical with the growing use of large-scale artificial intelligence models. One possible way to improve the performance per watt is to use mobile computing devices. The COUNTER framework [3] was proposed for sustainable resource management in cloud environments and focuses on GPU clusters to address efficiency challenges. The use of energy-efficient mobile

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

GPUs can further improve such workloads in this context. For example, modern biometry-related technologies achieve higher recognition accuracy [4] at the cost of increased computational demand. By offloading preprocessing tasks and parts of neural network inference to mobile devices, systems can reduce energy consumption while using additional distributed computing power. The authors of the TinyML framework [5] also note that increasing power consumption increases operational costs, contributes to carbon emissions, and creates constraints for scalable IoT systems. Involving mobile devices may offer a balanced solution between conventional datacenters and in-place computing, but standardized benchmarks must ensure fair evaluation and wider adoption.

This work [6] dissects how edge and mobile GPUs actually spend power, from single PTX instructions up to a real general matrix multiply workload. However, it lacks coverage of conventional datacenter hardware, which prevents comparisons of energy efficiency between mobile and datacenter computing.

In a previous study [7], we faced a methodological problem: although a unified model is viable, it is difficult to extend, and we are not the first to encounter this issue. Prabhakar Ragde [8] stated that mathematics leverages ambiguous notations and oversimplifications, which becomes a problem when translating theoretical models into practical computational implementations. As demonstrated by Matilde Gargiani [9], classical evaluation methods frequently fail to effectively scale to practical problems.

1.2. State of the Art

Some researchers use commercial application development experience to adjust their modeling approaches. For example, one study [10] explored the application of the Entity-Component Architecture (ECA) as a software design pattern for developing scalable and energy-efficient IoT brokers. Another study [11] investigated the application of ECA in the educational process.

It is known that ECA originates from game engines and simulators. However, in this article [12], the authors propose a model for a hierarchical architecture with worker nodes (vehicles). It can even be used to simulate military operations simulation [13]. In another case [14] ECA is used for real-world physics simulation. A survey of real-world frameworks [15] demonstrates that ECA better leverages safe concurrency, reduces contention, and broadens applicability beyond its traditional domains.

The study of machine learning-driven methodologies for energy consumption [16] identifies a rising need to quantify and optimize the energy usage of software running on portable devices. In the study "Evaluating the Energy Efficiency of Popular US Smartphone Health Care Apps" [17], the authors demonstrate that energy usage is not uniform but varies significantly depending on the intensity and subsystem type. Among mobile device subsystems, networking significantly contributes to energy inefficiencies [18].

One of the most influential architectural factors [19] is payload distribution between computing nodes. It has been shown that architectures that enable dynamic workload distribution have significantly lower energy profiles.

This study [20] evaluated whether everyday smartphone clusters can deliver real-time edge AI for computer vision. The results show that clusters of 3–4 phones often approach real-time and rival or surpass SBC setups in energy efficiency, especially when used with a pull-based scheduler.

Even under low usage levels, conventional modern servers consume a significant portion of their peak power [21], resulting in substantial energy losses. To properly account for this, the model should also simulate idle components and idle computers. A known model [22] considers both static and traffic-proportional energy use across different computing network elements but does not consider computing hardware. As a result, existing approaches only partially capture aspects of system energy consumption. Multiple factors must be considered to realistically assess the energy consumption of distributed computing systems: transmission energy, heat dissipation losses in power amplifiers, and the power usage of other RF-chain components [22]. This study [23] investigates how energy consumption in embedded and mobile devices varies depending on the transport-level protocol used to communicate with cloud or edge services. In this study, we aim to isolate the power consumption of different components as much as possible, following the approaches of other researchers [24].

1.3. Aim and approach

This study aims to improve the existing methodology for energy consumption forecasting in distributed computing systems. The shortcomings of traditional algebraic models, which struggle to incorporate the complexity of modern computing environments, are the first problem we address. Our approach is based on the Entity-Component Architecture (ECA): the distributed computing system is represented with Entities (each computer, router, or other node in a network), their subsystems are represented with Components (CPU, GPU, network, cooling system), and the behavior of each component has an algebraic representation. Another aim of this study is to address a

limitation of earlier research [7], where the same computing task was used for both benchmarking and model validation, i.e., a single algorithm was employed to benchmark the hardware and evaluate the energy model's prediction accuracy. In contrast, this study separates these two stages: hardware benchmarking is performed using one dedicated workload, whereas model validation is performed using a different workload. Validation is conducted on stationary and mobile computing networks (SCN and MCN, respectively).

Description of various Components, their roles and behavior, and distributed computing system schemas are provided in subsections 2.1-2.2. The approach for consumer power accounting is given in subsection 2.3, and the simulation algorithm is described in subsection 2.4.

Section 3 describes the end-to-end experiment descriptions: subsection 3.1 describes the configuration of a model of distributed computing systems, subsection 3.2 builds the model of a computing job, and subsection 3.3 provides simulation and measurement results.

The validation of the experiment is provided in Section 4. Here, we discuss the forecast deviation. In the Conclusion section, we summarize the results, highlight the approach's key strengths and weaknesses, and outline improvements for future research.

2. A Method of Power Consumption Evaluation

2.1. Power consumption measurement using a unified model of energy consumption

Our methodology focused on assessing indirect energy consumption associated with computational processes in previous research. We relied on internal hardware-level sensors to estimate energy use. Although this approach provided valuable insights into the energy efficiency of computing components under various workloads, its scope was limited.

The study was built around developing an improved unified model that follows the following criteria: it should incorporate idle hardware consumption and network infrastructure consumption, should not depend as heavily on the similarity of benchmarking and actual running tasks, should consider the topology of the network to incorporate consumption of all network nodes involved in data transmission, and the model should be easy to extend with new capabilities, yet it should provide reliable and useful results.

To enable subsystem-level assessment, we designed and executed a set of benchmarking procedures that specifically target individual components of interest: CPU, GPU and network interface. The benchmark was designed to maximize the load on a single subsystem while minimizing the interference from other subsystems. For CPU and GPU benchmarking, tensor-based deep learning inference workloads were used.

The energy consumption of the network interfaces was assessed through data transmission tests. These included large file transfers over the TCP/IP protocol and simulation of real-time communication scenarios. The benchmarking approach was built by subtracting idle consumption from consumption under load.

To ensure accurate readings and eliminate the limitations of internal sensors, all measurements were conducted using external power monitoring devices, particularly for network interfaces and storage devices that lack built-in energy reporting capabilities. This was a smart plug with instantaneous and cumulative consumption measurements for stationary servers. For mobile and network devices, we used a direct-current laboratory power source that also measures momentary and cumulative consumption. The transition from internal telemetry to dedicated external measurement devices marks a significant advancement, as we are confident that we are covering all consumption.

2.2. Mathematical representation of the power consumption unified model

In our previous research, we relied on standard algebraic mathematical modeling approaches to simulate and analyze energy consumption within computational systems. As the complexity of our studies increased, we encountered a significant increase in the complexity of algebraic modeling. Specifically, adding new variables resulted in nonlinear complexity growth, necessitating revisions in the approach. We decided to move to the Entity-Component Architecture (ECA). This architectural choice allows us to represent each node of our computing network as an entity and populate it with components representing a particular subsystem.

In our previous unified model [7], the energy consumption was represented as a system of three independent sub-models (1).

$$\mathrm{M} = \langle \mathrm{M_C}; \mathrm{M_J}; \mathrm{M_S} \rangle \quad (1)$$

where $\mathrm{M_C}$ is a computing system model;

$\mathrm{M_J}$ is a model of a computing job;

$\mathrm{M_S}$ is a model of the task distribution strategy.

The computational job model was represented as a collection of data chunks (artifacts) and an associated directed acyclic graph that indicates the dependencies and sequence of computation among the artifacts. This fundamental approach is maintained in the updated unified model of energy consumption, preserving the graph-based structure of task dependencies and artifact relationships. However, it introduces enhanced granularity by representing computational tasks as

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

sequences of stages that explicitly delineate CPU and GPU workloads rather than flat numeric metrics:

$$M_J = \langle T_i; A_i; D_{ij}; i, j = 0..J-1 \rangle, \quad (2)$$

where A_i is the amount of data contained in an artifact

D_{ij} is a matrix denoting the dependencies between tasks and artifacts;

J is the number of computing tasks in a computing job.

T_i is the computation complexity matrix.

Each row of the computational complexity matrix represents the corresponding complexity of a payload for a particular computing unit type, and each column represents a chunk of a computing task:

$$T_i = \begin{bmatrix} TP_{0,GPU} & \dots & TP_{TC_i-1,GPU} \\ TP_{0,CPU} & \dots & TP_{TC_i-1,CPU} \end{bmatrix}, \quad (3)$$

where $TP_{k,CPU}$ and $TP_{k,GPU}$ represent the computational complexity of the i-th task chunk, respectively.

TC_i is the number of task chunks in the i-th computational task.

By decomposing computational tasks into discrete CPU and GPU components, we can capture the energy consumption of different computing node subsystems and decompose the computation into chunks. Representing a computation as a series of chunks allows us to represent a non-homogeneous payload, for example: decompress an image on CPU, process it on CPU and GPU, and compress it on CPU. The computing system model is refined and represented as a set of task chunks as follows:

$$M_C = \langle E_i; i = 0..C-1 \rangle, \quad (4)$$

where E_i is an Entity representation based on the ECA approach.

Here, an Entity represents a computing system's node. Previously, it was only a node with an actual computing unit; now, it can represent network hardware, such as switches or routers, by adding corresponding components. An Entity can be represented as a vector of components as follows:

$$E_i = [EC_0, \dots, EC_{CC_i}], \quad (5)$$

where EC_k represents the components;

CC_i is the number of components in the i-th entity.

Each component is an abstraction with its own implementation and behavior. Adding a new component adds a capability to an entity, and removing a component removes the capability from the entity without refining other components. This is how the ECA approach works and grants extensibility and scalability. Each component can be defined as follows:

$$EC = \langle Type, Owner, Behavior \rangle, \quad (6)$$

where Type is a unique identifier that describes the type of a particular Component;

Owner is information about the owning Entity;

Behavior is the actual behavior of a component.

This representation is not algebraic, but it is logical and is more common in computer architecture. From a computer science point of view, a Component is a polymorphic entity that knows the Entity that owns it, its Type, and can perform abstract behavior. A particular Component representing a specific subsystem (e.g., CPU, GPU, network interface) of a computing node can be represented by assigning a specific Behavior. With this representation, we can define the following abstract simulation execution operation:

$$ExecutionOperation(M_C; Type), \quad (7)$$

where MC is the computing system model;

Type is a unique identifier of a component type.

The execution operation enumerates through all Components of all Entities in a computing system model and triggers the execution of a Behavior of each Component having the specified Type.

The model of computing tasks distribution strategy M_S is used "as is" from our previous work and is not a subject of this research. The strategy is responsible for the assignment of a Task to a computing node. It can be completely random, with the only requirement that it should be the same in all simulations. The resulting consumed energy can be defined as a function of M:

$$E(M) = E(M_C; M_J; M_S). \quad (8)$$

Let us go through the Components that we use. The Chassis component regulates Node availability through a Markov process, where a reliable Chassis has a zero probability of transitioning to an off state and an unreliable chassis exhibits a defined non-zero probability per tick. Thus, the state of a Node (available or disabled) is determined probabilistically per tick:

$$P_{on \to off} = p, P_{off \to on} = q, \quad (9)$$

where p and q are the transition probabilities between on and off states, respectively.

The Control Center (CC) Component acts as the central management hub of the distributed computing network, coordinating communication and tasks. Each computational network has only one CC Node, which cannot serve as a computational participant (i.e., it cannot

have an Agent).

Computational capabilities are represented by Processing Unit (PU) components of a particular type (CPU or GPU), with each node being limited to one PU component of each type, symbolizing their respective computational capacities and specialized workloads. PUs share a consistent operational model in which each possesses a defined maximum operational capacity, which is expressed as the maximum number of operations per second (ops/sec). During each simulation tick, these components are assigned computational tasks. The maximum capability of the respective components limits the actual processing volume per tick, subsequently translating into the momentary load used for energy calculations:

$$L(t) = \frac{Ops(t)}{Ops_{max}}, \qquad (10)$$

where t – the number of ticks in a simulation

Ops(t) – the number of operations assigned to a PU on a particular tick.

Ops_{max} – the maximum number of operations that the PU can perform per simulation tick.

Data Transfer Interfaces (DTIs) facilitate all communication tasks and come in various types, such as wired LAN, WiFi, and USB. DTIs enable connectivity exclusively between interfaces of the same type on different nodes. Let us designate a DTI of a particular type as DTI/Type: DTI/wiredLAN, DTI/wirelessLAN, DTI/USB, etc. Since a DTI is, by definition, an interface, it has 2 sides that it interfaces with. For clarity, let us call them the interface side (or i-side) and the backend side (or b-side). The interface side of a DTI interacts with another DTI of the same type, while the backend side is connected to a controller. For example, in the case of DTI/wiredLAN, the interface side will be an RJ45 port, and the backend side will be soldered on the circuit. A DTI is a bi-directional graph edge with a cost representing energy consumption based on traffic.

A component capable of simulating various topology cases is needed to simulate a network. The Relay component joins b-sides of multiple DTIs of identical types using star topology (see Figure 1).

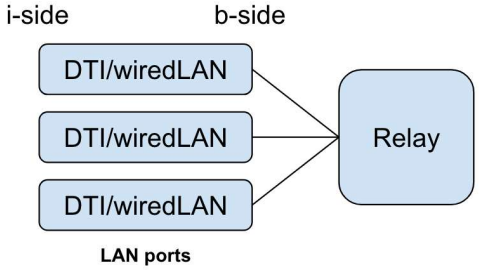

Fig 1. Example of Relay Component usage to build Entity representing Switch node

DTIs joined to the same Relay can relay traffic between each other. For example, a LAN switch is represented as many DTIs (one per physical port) and a Relay. The Bridge component represents a special case of bidirectional graph edge connectivity by linking i-sides of two DTIs of the same type, thereby enabling seamless communication across network segments. For example, a Bridge can connect the DTI/wiredLAN's of 2 groups of DTIs joined by a Relay. In this case, it simulates a regular router. Figure 2 shows the scheme.

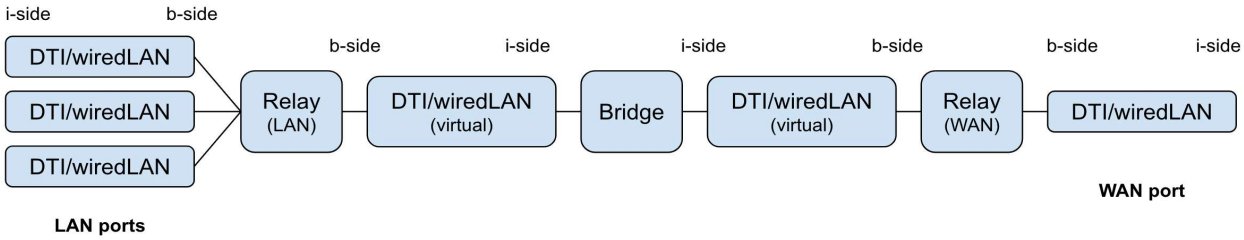

Fig 2. Structure of Entity representing Router node built of Components

The Proxy component serves as another special case of a bidirectional graph edge facilitating data transfer between distinct DTIs within a node, supporting scenarios where, for example, a Node with a DTI/USB interface can only use another Node having both DTI/USB and DTI/LAN to access a particular LAN. Unlike the Bridge, the Proxy connects b-sides of 2 DTIs of any type.

DTIs are characterized by their maximum throughput capacity, which is defined by the bandwidth (max bytes per second). The DTI load can be computed in a similar way, but based on the used bandwidth:

$$L(t) = \frac{Bytes(t)}{Bytes_{max}}, \qquad (11)$$

where t – the number of ticks in a simulation

Bytes(t) – the number of bytes planned for a DTI transfer at a particular tick;

$Bytes_{max}$ – the maximum number of bytes that the DTI

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

can transfer per simulation tick.

DTIs communicate exclusively with matching-type DTIs, thereby enforcing compatibility restrictions. When two DTIs establish a connection, the slower DTI constricts the effective bandwidth. Service components such as Relay, Bridge, and Proxy alter network topology by interconnecting DTIs. The bandwidth through these components is restricted to the lowest available bandwidth among all connected DTIs, by a Relay with multiple DTIs sharing bandwidth constraints:

$$\text{Bytes} = \min(\text{Bytes}_{\text{DTI1}}, \dots, \text{Bytes}_{\text{DTIN}}). \quad (12)$$

The Agent component encapsulates the behavior of a computational software that executes computing tasks, handling task scopes, initiation, and data artifact exchanges. Only nodes equipped with Agents can participate in computational activities. Agents do not directly contribute to power consumption, but they control how PUs and DTIs function.

The Idle component represents a static consumer of Node components, independent of their activity and computational load.

The power consumption prediction model for our computing network employs a discrete, time-based simulation with fixed, tick-based intervals. Each Entity within the model primarily serves as a stateless container without embedded operational logic. Each Consumer within the simulation is structured to evaluate consumption based on the component load. At every tick interval, the owning component sends its current load on its Consumer, prompting the calculation of the energy consumed based on predefined load-consumption relationships. Momentary consumption C can be calculated as follows:

$$C(L) = C_{idle} + (C_{max} - C_{idle})L, \quad (13)$$

where L – momentary load;

C_{idle} – idle consumption;

C_{max} – max consumption;

To calculate cumulative consumption within a simulated time, we simply sum the momentary consumption at every tick.

$$C(L, t_{start}, t_{end}) = \sum_{t=t_{start}}^{t_{end}} C(L(t))dt, \quad (14)$$

where t – the current tick number

t_{start} – the first tick of a simulated period;

t_{end} – the last tick of a simulated period;

dt – the simulation tick duration;

L(t) – the load at a particular tick of a simulation;

Figure 3 shows the structure of the Stationary Computing Network that we are using in the experiment: 2 identical PCs with a CPU, GPU, and wired network interface, a Router connecting the PCs to the Internet, and the cloud-deployed CC. The CC and the Internet do not have CPUs and Idle consumption, and we exclude them from the equation. The Router doesn't have a CPU because it is not supposed to run computation tasks, and its controller's power usage is associated with DTIs.

Figure 4 shows the structure of a Mobile Computing Network. It is completely local and composed of 3 mobile phones with CPU, GPU, and USB interfaces (for simplicity, we consider Lightning as USB). The Raspberry Pi 5 runs CC and is responsible for data exchange.

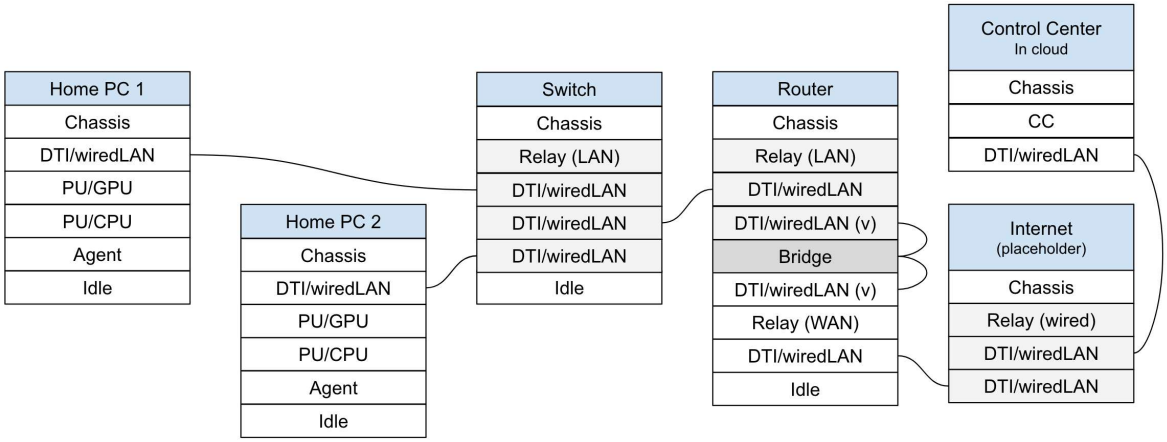

Fig 3. Structure of a Stationary Computing System

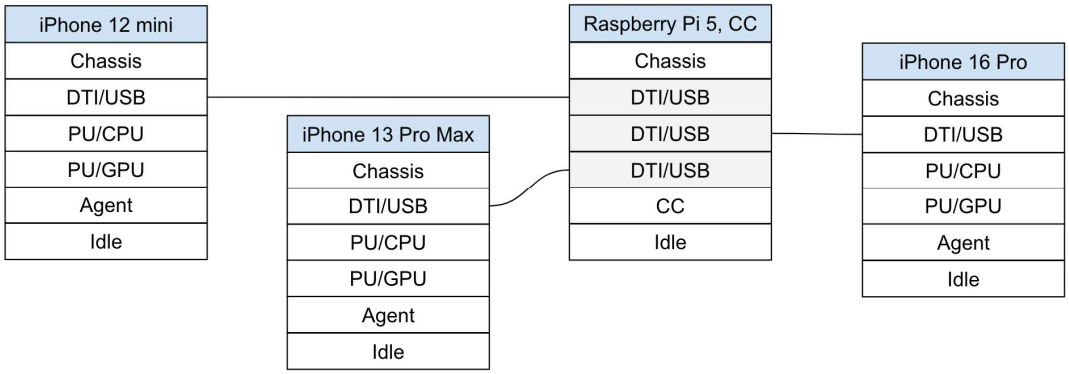

Fig 4. Structure of a Mobile Computing System

The topology components (Relay, Bridge, Proxy) describe how data is transferred inside a node, whereas the connections between DTIs describe how data flows between the nodes (Figures 1 and 2).

Nodes that we do not control or simulate (Internet, cloud hosting) may not have power-consuming Components, but they can have topological components to simulate availability using Chassis Component (9).

2.3 Energy consumption simulation using the power consumption unified model

To run the simulation, we need to build nodes as Entities, fill them with Components, assign unique identifiers to each Component and establish connections between nodes using DTI interfaces. We also need to initialize the key-value storage and the simulation tick duration. A tick is a discrete simulation step that simulates a fixed virtual time in a computing system. Once the above is completed, the simulation occurs through iterative steps defined within each tick:

1. Chassis State Step or Execution Operation for the Component-type Chassis. If a Node becomes unavailable, its Agent is reset, and its associated data transfers and computations are halted.

2. Nodes validate or reconstruct communication paths to the Control Center (CC). This step assumes an Execution Operation for the following Component Types: Relay, Bridge, Proxy.

3. Agent Operation Step. Agents initiate data transfer operations, prioritizing uploads over downloads when necessary. Transfer speeds across DTIs are proportionally adjusted if the transfer speed exceeds the maximum bandwidth. This is achieved through the Execution Operation of the Agent Component type.

4. Operation and Load Calculation Step. In this step, we invoke the Execution Operation for all DTIs, PUs, and Idles. The respective load values are set using formulas (10) and (11). These momentary loads are sent to the nested Consumers, which can calculate consumption using Equation (13). For simplicity, the dependency between load and consumption is considered to be linear (9).

5. Consumption Aggregation Step. We enumerate each Component of each Entity and aggregate all momentary consumption of each Consumer available in the computing system. At this stage, we use key-value storage, where the key is a composite of the Entity and Component's unique identifiers, and the value is the accumulated consumption in joules. Because we have the Consumer's momentary consumption and the simulation's tick duration, we simply add their products to the value in the key-value storage.

6. Compute the job update step. The system updates the status of all computing Tasks in the computing system. The simulation ends if all artifacts are uploaded to the CC.

Each Consumer is tied explicitly to a Component and, subsequently, each Component to an Entity-Node. After the simulation completes, the key-value store contains the consumption of each Component. The sum of all values in a key-value storage gives the simulated consumption. This is how the resulting energy consumption is calculated as follows (8).

2.4. Summarizing the power consumption estimation method

The unified energy consumption model described above is an essential part of power consumption estimation, but the model itself is not an end-to-end solution. The end-to-end solution has 4 main steps.

First, we measure the consumption of each component of a computing system (e.g., DTIs, CPUs, GPUs, idle consumption). In our study, power consumption is measured directly using external tools (a smart plug and a laboratory power source, both capable of measuring cumulative power consumption). Second, we build a distributed computing network model using the values from the first step. Third, we build a computing task model. We build a graph-like computing

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

job model: the complexity of a Task is set not by a flat number (as in our previous research), but by a matrix representing the computational complexity of different steps using different computing unit types (3). This data is coded into a matrix where each column represents a step and each row represents the load for a particular computing unit type. This allows us to simulate Task computation on specific hardware and account for the consumption of the idling computing units. The final fourth step is to apply the unified energy consumption model and forecast energy consumption.

3. Experimental evaluation

3.1. Measurement of the hardware power consumption

We conducted controlled measurements involving three Apple smartphone models: iPhone 12 mini, iPhone 13 Pro Max, and iPhone 16 Pro (Figure 4) to isolate and quantify the energy consumption attributable specifically to USB/Lightning data transfer on modern Apple smartphones. The goal was to eliminate contributions from internal power sources, such as accumulators, and from unrelated subsystems, including radio and display circuits, which was a significant flaw in our previous paper on the estimation of power consumption in mobile devices for cloud computing [26]. Each device was fully charged, set to Airplane mode, and connected via USB to a Raspberry Pi 5 powered by a constant-voltage power supply in the laboratory. The power supply concurrently acted as an energy integrator, recording cumulative power consumption across one-hour intervals. We used a smart plug with cumulative consumed power measurement capabilities to measure stationary servers. The declared deviation is ±1%. For mobile and network hardware, we used a laboratory power supply unit capable of doing the same with a declared deviation of <1%. For each test, it was at least 3 runs of 1 h each with the average time taken. Every device was preliminarily warmed up (to ensure that temperature, cooling, and battery charge stabilize). All logging was delegated to the power supply and smart plug. We measured four conditions for each device: Raspberry Pi idle (no phone connected), Raspberry Pi with a connected phone in an idle state, Raspberry Pi with a connected phone and active screen, and finally, during active data transfer between the phone and Raspberry Pi. The measurements are presented in Table 1. From left to right, columns meaning: phone model name, the measured idle Raspberry Pi 5 power consumption while connected to a display, the measured idle Raspberry Pi 5 + idle phone consumption, the calculated pure idle phone consumption (difference between columns 4 and 5), the measured consumption of RPi5 + non-loaded phone with the unlocked screen, and the measured RPi5 + phone with the unlocked screen and transferring data via a wired interface. The last, 7th column is the calculated pure energy consumption while transferring data via the wired interface.

The “Phone Idle” column is the difference between “RPi5 Idle” and “RPi5+Phone Idle”, which includes RPi5’s USB consumption, mobile phone USB-C/Lightning consumption, and mobile phone Idle consumption. We cannot distinguish these 3 things hence “RPi5 Idle” is accounted for as the idle consumption of the mobile device. The idle consumption of Raspberry Pi 5 without any connected peripheral devices is 2.72W (in Table 1, we measured consumption as 4.03W, the difference is caused by connecting a high-resolution display).

Table 1

Smartphones’ DTIs consumption under load measurements

	RPi5 Idle (W)	RPi5+phone Idle (W)	Phone Idle (W)	Total Screen On (W)	Total Data tr. (W)	Interfaces Data tr. (W)
iPhone 12 mini	4.03	4.48	0.45	5.22	6.71	1.49
iPhone 13 Pro Max		4.56	0.53	5.1	6.37	1.27
iPhone 16 Pro		4.5	0.47	5.04	6.11	1.07

Table 2

Smartphones’ DTIs bandwidth measurements

	Average Data tr. Speed (MB/s)	Interfaces Data tr. (W)	Rate MB/J
iPhone 12 mini	28.2	1.49	18.92
iPhone 13 Pro Max	30.2	1.27	23.77
iPhone 16 Pro	29.8	1.07	27.85

"Total Screen On" is the total power consumption of the RPi5 and the phone when the screen is unlocked (screen brightness is set to minimum). "Total Data transfer" is the total consumption while the screen is unlocked and the data is being transferred. "Interfaces Data transfer" is the difference between the previous two columns and reflects the total power consumption of both the smartphone and RPi5 while the data is being transferred.

"Average data transfer speed" is about 30Mb/s, which is lower than the expected peak of 60MB/s, which is expected by Apple devices via cables, but this is likely related to issues with Raspbian's USB drivers and Apple devices support. The results are provided in Table 2. Column descriptions from left to right: phone model name, measured average data transfer speed over the measured period, and the calculated pure energy consumption of data transfer, taken from Table 1 column 7. The 4th column is the calculated data transfer energy effort, calculated as column 2 divided by column 3.

To quantify the isolated energy consumption of CPU and GPU subsystems in modern smartphones, we conducted controlled measurements on the iPhone 12 mini, iPhone 13 Pro Max, and iPhone 16 Pro. The benchmark, developed in-house, involved repeated inference of MobileNetV2 and its 25% pruned variant, corresponding to approximately 310 MFLOPs and 275 MFLOPs per inference, respectively. On each device, 10,000 inferences were executed on the CPU and 100,000 on the GPU, allowing for precise estimation of floating-point operations per second (FLOPS) under known workloads (Table 3). Column descriptions from left to right: phone model name, measured execution time of 10000 inferences of full MobileNetV2, calculated performance while running full MobileNetV2 evaluated as the inference complexity (310 MFLOPs) multiplied by the number of runs (10000) and divided by the execution time (column 2 value), measured execution time of the 25% pruned MobileNetV2, calculated performance while running the 25% pruned MobileNetV2 evaluated as the inference complexity (275 MFLOPs) multiplied by the number of runs (10000) and divided by the execution time (column 4 value), and the last column is an averaged performance value (average between columns 3 and 5). Simultaneously, energy consumption was recorded using a laboratory power source, aggregating power consumption over time (Table 4). To measure stable consumption over time, we ran the device in idle mode with the screen on for 1 h, then conducted 3 benchmark runs per device to avoid side effects. Table 4 columns from left to right: phone model name, calculated phone consumption with the screen on (by subtracting the idle RPi5 consumption of 2.72 W from the direct measurement), calculated consumption while running MobileNetV2 inference, calculated CPU-only consumption as the difference between the 2 previous columns, and calculated operations-per-joule ratio as the previous column divided by the value from the last column of Table 3 for the corresponding phone model name

The same procedure was applied for GPU - we ran the same net on the same library but with the CoreML backend (Table 3). We observed a 10-12x execution speed boost for the CoreML GPU backend compared to the CPU backend. Note that we run 10 times as many inferences per benchmark run on the GPU. Column 5 of Table 4 shows that, from an energy perspective, the GPU outperforms the CPU by 16-18 times.

Table 3

Smartphones CPU and GPU performance measurements

	Full MobileNetV2, 310 MFLOPS		Pruned 75% MobileNetV2, 275 MFLOPS		Average
	10k Inference Time (s)	Perf. (GFLOP/S)	10k Inference Time (s)	Perf. (GFLOP/S)	Perf. (GFLOP/S)
CPU measurements					
iPhone 12 mini	116.2	26.68	106	25.94	26.31
iPhone 13 Pro Max	102	30.39	91.2	30.15	30.27
iPhone 16 Pro	80.8	38.37	72.5	37.93	38.15
GPUs measurements					
iPhone 12 mini	114.1	271.7	106	259.4	265.6
iPhone 13 Pro Max	95.9	323.3	88.2	311.8	317.6
iPhone 16 Pro	66.1	469	60.2	456.8	462.9

 ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

Table 4

Smartphones CPU and GPU consumption measurements under a load

	Idle (screen on) consumption (W)	MobileNetV2 consumption (W)	PU-ony consumption (W)	PU efficiency (GFLOPS/J)
CPU measurements				
iPhone 12 mini	0.93	7.79	6.86	3.83
iPhone 13 Pro Max	1.21	8.47	7.26	4.17
iPhone 16 Pro	1.1	8.24	7.14	5.34
GPUs measurements				
iPhone 12 mini	1.17	5.05	3.88	68.4
iPhone 13 Pro Max	1.27	5.78	4.51	70.4
iPhone 16 Pro	1.11	6.36	5.25	88.2

The same benchmarking methodology was applied to a high-performance stationary computing system comprising an Intel Core i9-10850K CPU, an NVIDIA RTX 3090 GPU, and a 1Gb LAN interface (Figure 3).

The goal remained consistent: to isolate and quantify each component's power consumption.

The benchmark procedure mirrored that described earlier: it used MobileNetV2 inference and its 25% pruned variant, corresponding to approximately 310 and 275 MFLOPs per inference, respectively. For both CPU and GPU measurements, the MobileNetV2 and its pruned version were executed 100,000 times using TensorFlow on CPU and GPU, respectively (Table 5). The structure of Table 5 is similar to Table 3: hardware model name, inference execution duration for full MobileNetV2, calculated performance in floating-point operations per second (inference complexity multiplied by the number of runs and divided by execution time from column 2), calculated performance and average performance between columns 3 and 5: inference execution duration for the 25% pruned MobileNetV2. Power consumption was captured by a smart plug power meter attached directly to the system's power lines throughout both scenarios, providing integrated energy data over the full duration of each test (Table 6). The content of Table 6 is identical to that of Table 4: we measure hardware consumption under load and calculate the efficiency of floating-point operations per joule of energy. The effective computational throughput was calculated in FLOPS from the execution time and known FLOP count (see column PU efficiency in Table 6). In parallel, the energy efficiency was evaluated by dividing the total operations by the total energy consumed (in joules), yielding a measure in FLOPS per watt. We preserve the same MobileNetV2, inference loads, and power measurement procedures for MCN. It follows the normalized FLOP-per-watt metric as a unified principle for evaluating the efficiency of computation across heterogeneous environments. The same testbed was used to measure LAN Interface power consumption – we measured the idle consumption of the entire server, then enabled data transfer (average speed was 109 MB/s), and then measured energy consumption while data were being transferred (Table 6). The calculation principle is similar to that used for computing units: we measure idle system consumption, then measure consumption under load, then calculate pure data transfer consumption and data transfer energy efficiency as the ratio of data transferred per second to joules per second (watts).

Table 5

Stationary computer processing units' performance measurements

	Full MobileNetV2, 310 MFLOPS		Pruned 75% MobileNetV2, 275 MFLOPS		Average
	100k Inference Time (s)	Perf. (GFLOP/S)	100k Inference Time (s)	Perf. (GFLOP/S)	Perf. (GFLOP/S)
Core i9 10850K	259.7	119.4	234.5	117.3	118.4
RTX 3090	4.1	7650	3.66	7513	7581

Table 6

Stationary computer hardware consumption under load measurements

	Idle consumption (W)	MobileNetV2 consumption (W)	PU-only or LAN-only consumption (W)	PU or data transfer efficiency
Core i9 10850K	107	222	115	1.03 GFLOPS/J
RTX 3090		365	258	29.4 GFLOPS/J
LAN Interface		110	3	36.3 MB/J

In our experiment there are 2 standalone devices (Figure 3): a 1Gb LAN Switch and a 1Gb WiFi Router. To measure consumption, we connected the Switch to a laboratory power source and then started connecting Ethernet cables one by one to measure the average consumption over a certain period of time (10-15 min). Measurements were taken for 0 to 3 connected cables in an idle state and under full load (all-to-all data transfer). The idle consumption of the LAN switch was 0.16 W, the average per-port idle consumption was 0.246 W, and the average per-port consumption under load was 0.326 W (Table 7). This behavior can be simulated with a router comprising static power consumers and DTIs with idle and maximum consumption joined by a Relay (Figure 1).

The scheme for the WiFi Router is pretty much the same. There were up to 3 connections and 3 series of measurements for each number of connections: idle consumption, local transfer (all-to-all data transfer at the maximum rate), and Internet data transfer (all connected devices loaded with large files from the Internet). The measurements (Table 8) show that the idle connections add an average of 0.055 W each to the consumption. Local transfer consumption averages 5.41 W, with a data transfer rate of 104.4 MB/s. Internet transfer consumption is 6.12 W at a rate of 43.93 MB/s. We can simulate such a router with an Idle consumption of 2.81W, DTIs with an idle and max consumption of 0.055W, a Relay joining them and consuming from 0 to 5.41W - 2.81W = 2.6W and the Bridge's DTI consuming from 0 to 6.12W - 5.41W = 0.71W (Figure 2).

Table 7

Measurements of the WAN switch consumption and transfer speed

	Idle (W)	Full load (W)	Transfer speed (MB/s)
Idle consumption	0.16		
1 port	0.38		
2 ports	0.64	0.7	112.8
3 ports	0.97	1.02	115.6

Table 8

WiFi Router consumption and measurement of transfer speed

	Idle (W)	Local transfer consumption (W)	Local transfer speed (MB/s)	Internet transfer consumption (W)	Internet transfer speed (MB/s)
Idle consumption	2.81				
1 connection	2.88			6.11	42.4
2 connections	2.82	5.51	107	5.91	43.9
3 connections	2.97	5.32	106.8	6.24	46.5

3.4. Computing task definition based on measurements

To validate our unified power consumption forecasting model, two distinct test cases were conducted using inference with the Inception V3 neural network. Input artifacts are 10000 RGB images of 299x299 pixels in size in JPEG format. The output artifacts are 10000 images with added text: a list of recognized objects in JPEG format. The computing Task has the following structure.

The average size of a 299x299 RGB image in JPEG format in our 10k image dataset is 87kb. The average computation complexity of JPEG compression was estimated to be 162 FLOPs per pixel, and that of decompression was estimated to be 108 FLOPs per pixel.

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

Tests were conducted on various images using libjpeg at 90% quality. Hence, we can evaluate decompression procedures as 9.6M FLOPs and compression as 14M FLOPs. According to the official source, the inference complexity of the Inception V3 tflite model is 6B FLOPs. The inscription complexity is negligible, and it can be ignored. Figure 5 shows the scheme of a single computing task.

The entire computing job is to perform inference on 10000 unique images, totaling 60 trillion FLOPs of CPU/GPU load and 236 billion FLOPs of CPU-only load.

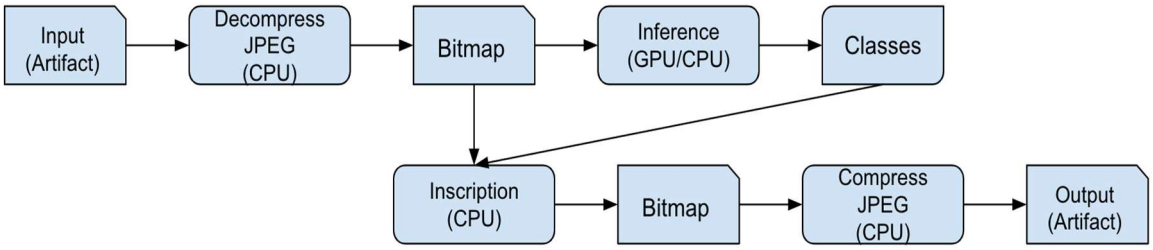

Fig 5. Structure of the computation task

3.5. Experimental data and simulation result

Experiments are conducted on 2 setups: a stationary computing network (SCN) and a mobile computing Network (MCN). On both setups, computing was conducted using CPU+GPU and CPU-only modes. The SCN setup involved two stationary servers, each equipped with an Intel Core i9-10850K CPU and an NVIDIA RTX 3090 GPU, interconnected via a LAN switch (Figure 3). The centralized Control Center (CC) was hosted remotely (in the cloud) and accessed through a Router also connected to the same LAN switch. The MCN scenario involved three smartphones (iPhone 12 mini, iPhone 13 Pro Max, and iPhone 16 Pro) connected via USB to a Raspberry Pi 5 hosting the CC (Figure 4). The smartphones were operated in parallel.

During the SCN run, we can directly measure the consumption of each node (using multiple plug meters), and for the MCN, we can measure only the total system consumption (each node is powered by the Raspberry Pi or the Raspberry Pi). The results of the direct measurements are provided in columns 4 and 5 (Table 9). The SCN CPU-only run took 274 s, and the CPU+GPU run took 12 s. The MCN CPU-only run took 662 s, and the CPU+CPU run took 71 s. The SCN simulation predicted a CPU-only run time of 255 s and a CPU+GPU run time of 8 s. The MCN simulation time was 639 s for the CPU-only and 58 s for the CPU+GPU run.

The simulation results are provided in columns 2 and 3 (Table 9). For simulated consumption, we provide separate results of idle, CPU, GPU, and LAN component consumption for each server. However, we need to sum all the power consumed per server to compare them to real measurements. For example, we can take the measured CPU-only consumption of server 1 from column 4, row 3 and compare it to the sum of the simulated CPU-only consumption of Idle, CPU, GPU, and LAN values of server 1 from column 2, rows 3-6 (Table 9). Additional details of the simulation are added only to demonstrate that per-component power consumption simulation can be obtained, which may be useful for future research.

Further discussion will mostly refer to the “Total SCN” and “Total MCN” rows, specifically comparing column 2 with column 4 (CPU-only runs) and column 3 with column 5 (CPU+GPU runs).

Table 9

Simulated and measured energy consumption

Component	CPU-only, simulated (J)	CPU+GPU, simulated (J)	CPU-only, measured (J)	CPU+GPU, measured (J)
SCN (Stationary Computing Network)				
Server 1 Idle	27248	842	59062	3768
Server 1 CPU	29955	228		
Server 1 GPU	0	2034		
Server 1 LAN	26	26		

Component	CPU-only, simulated (J)	CPU+GPU, simulated (J)	CPU-only, measured (J)	CPU+GPU, measured (J)
Server 2 Idle	27248	842	59 669	4004
Server 2 CPU	30101	227		
Server 2 GPU	0	2030		
Server 2 LAN	27	26		
LAN Switch	9	1	271	15
Router	115	4	1065	58
Total SCN	**114729**	**6260**	**120067**	**7845**
MCN (Mobile Computing Network)				
Raspberry Pi Idle	1739	158	17982	1491
iPhone 12 mini Lightning	67	66		
iPhone 12 mini Idle	593	54		
iPhone 12 mini CPU	4338	18		
iPhone 12 mini GPU	0	218		
iPhone 13 Pro Max Lightning	56	56		
iPhone 13 Pro Max Idle	774	70		
iPhone 13 Pro Max CPU	4578	19		
iPhone 13 Pro Max GPU	0	255		
iPhone 16 Pro Type-C	48	48		
iPhone 16 Pro Idle	702	64		
iPhone 16 Pro CPU	4503	19		
iPhone 16 Pro GPU	0	296		
Total MCN	**17344**	**1341**	**17982**	**1491**

4. Discussion

In the SCN case, the simulated energy consumption for the CPU-only mode totaled 114729 J, compared with an experimentally measured value of 120067 J (Table 9), resulting in a deviation of 4.4%. In the CPU+GPU mode, the simulation yielded a total of 6260 J, while the measured total was 7845 J, resulting in a 20.2% deviation. This deviation in CPU + GPU mode is due to flaws in the computing system, leading to an underloaded GPU. The flat performance of the GPU allows it to finish computing within 8 s, whereas the real run took 12 s. While the GPU may be underloaded and consume less, the server's idle consumption remains the same. Presumably, the non-linear scaling of the consumed power under partial load also contributes to this study and will be a subject of further research. However, this shows that underloading conventional stationary servers worsens the FLOPs/J ratio in favor of mobile hardware.

For the MCN configuration, which featured three iPhones connected to a Raspberry Pi 5 via USB, the simulated consumption for the CPU-only run totaled 17344 J, which was 3.5% less than the measured total value of 17982 J (Table 9). In the CPU+GPU mode, the simulation yielded 1341 J versus a measured 1491 J, with a 10.1% deviation. The nature of the deviation seems to be similar to that of the SCN, but the impact is much lower due to the lower idle consumption of mobile platforms.

The experimental results confirm the accuracy and applicability of the proposed ECA-based unified model of power consumption and the simulation conducted using it. We were able to add new capabilities without refining or touching existing components. The model was evaluated on both stationary (SCN) and mobile (MCN) computing networks in CPU-only and CPU+GPU modes.

Overall, the simulation model captures energy usage trends with high fidelity across architectures and computational profiles. Deviations, particularly for GPU-intensive tasks that partially load GPUs, suggest that future improvements should incorporate thermal throttling and hardware-specific scheduling, as well as account for service routines in real software that lead to

higher task execution times than in the ideal case. The current unified model of energy consumption is sufficiently robust for scenario analysis of which tasks can be offloaded to a mobile device-based computing systems.

5. Conclusions

The research presented demonstrates a significant advancement in modeling and accurately forecasting power consumption within distributed computing networks by adopting the ECA approach. Our approach successfully addressed the prior methodological limitations inherent to traditional algebraic frameworks, specifically enhancing modularity, scalability, and comprehensibility through the explicit isolation of energy-consuming components, such as CPUs, GPUs, data transmission interfaces, and network infrastructure. First, unlike the previous paper, we eliminated the use of the same computing payload in computing system benchmarking and real computing, which makes our approach more generic. Second, we transform a clunky monolith math model into a scalable ECA approach, which allows us to easily cover multiple aspects of distributed computing networks. The third major achievement of this paper is that we moved away from sensor-based measurement of the consumption of specific components and started using an all-in approach to measuring overall consumption. This may reduce accuracy at some point, but it considers all equipment, including PSU power draws, cooling systems, and some idle consumption.

Empirical validation of both SCN and MCN confirms the high fidelity of our simulation model. For SCN, the CPU-only mode showed a minimal deviation (4.4%) between the simulated and experimentally measured power consumption. Conversely, the CPU+GPU scenario exhibited a larger discrepancy of approximately 20.2%, primarily due to suboptimal GPU utilization, which disproportionately influenced total energy usage. This underscores two things: the critical influence of efficient resource utilization on energy efficiency in stationary server environments and an insufficient approach to computing task simulation. For MCN configurations involving smartphones, the results were notably precise, with only a 3.5% deviation in CPU-only mode and a manageable 10.1% deviation in CPU+GPU mode. This highlights another aspect: the sparse load of MCNs contributes much less to the resulting energy consumption.

The presented study reveals areas for further improvement. First, the model uses linear interpolation based on the load between idle and maximum consumption. This relation is not linear. The current model neglects thermal throttling mechanisms and real-time adaptive hardware scheduling. Addressing these factors is essential for enhancing the precision of GPU-related simulations, particularly in high-performance hardware contexts. Additionally, measurements highlighted an unexpectedly low performance in USB data transfers between smartphones and Raspberry Pi devices. This did not affect our research because USB was not a bottleneck; however, resolving these issues would enhance the energy efficiency of USB data transfer.

Contribution of authors: methodology conceptualization – **Oleksandr Mamchych**; approach verification – **Maksym Volk**; task formulation, software development, experimental investigation, data analysis, and manuscript writing – **Oleksandr Mamchych**; manuscript review and supervision – **Maksym Volk**.

All authors have read and approved the published version of this manuscript.

Conflict of interest

The authors declare that they have no conflict of interest in relation to this research, whether financial, personal, authorship or otherwise, that could affect the research and its results presented in this paper.

Financing

The research was conducted without financial support.

Data availability

The manuscript has no associated data.

Use of artificial intelligence

The authors confirm that they did use artificial intelligence methods exclusively for spellchecking.

References

1. Shehabi, A., Newkirk, A., Smith, S. J., Hubbard, A., Lei, N., Siddik, Md A. B., Holecek, B., Koomey, J., Masanet, E., & Sartor, D. 2024 United States Data Center Energy Usage Report. *Lawrence Berkeley National Laboratory*, 2024, report LBNL-2001637. DOI: 10.71468/P1WC7Q

2. Suarez, E., Amaya, J., Frank, M., Freyermuth, O., Girone, M., Kostrzewa, B., & Pfalzner, S. Energy Efficiency trends in HPC: what high-energy and astrophysicists need to know. *Frontiers in Physics*, 2025, no. 13. DOI: 10.3389/fphy.2025.1542474

3. Wang, H., Gill, S., & Uhlig, S. COUNTER: Cluster GCN based Energy Efficient Resource Management for Sustainable Cloud Computing Environments. *45th IEEE International Conference on Distributed Computing Systems*, 2025. DOI: 10.48550/arXiv.2504.09995

4. Barkovska, O., Ruban, I., Romanenkov, Y., Botnar, P., & Havrashenko, A. Determining an approach to iris recognition depending on shooting conditions. *Eastern-European Journal of Enterprise Technologies*, 2025, vol. 134, no. 2, pp. 17–27. DOI: 10.15587/1729-4061.2025.325517

5. Elhanashi, A., Dini, P., Saponara, S., & Zheng, Q. Advancements in TinyML: Applications, Limitations, and Impact on IoT Devices. *Electronics*, 2024, vol. 13, article no. 3562. DOI: 10.3390/electronics13173562

6. Rafi, M. A., Chau, K., & Jeon, H. Uncovering Detailed Power Characterizations of GPUs on Edge Platforms. *Proceedings of the 17th Workshop on General Purpose Processing Using GPU (GPGPU 2025)*, 2025, vol. 17, pp. 48–54. DOI: 10.1145/3725798.3725806

7. Mamchych, O., & Volk, M. A unified model and method for forecasting energy consumption in distributed computing systems based on stationary and mobile devices. *Radioelectronic and Computer Systems*, 2024, no. 2, pp. 120–135. DOI: 10.32620/reks.2024.2.10

8. Prabhakar, R. Mathematics Is Imprecise. *Electronic Proceedings in Theoretical Computer Science*, 2013, vol. 106, pp. 40–49. DOI: 10.4204/EPTCS.106.3

9. Gargiani, M., Pawlowsky, P., Sieber, R., Hapla, V., & Lygeros, R. A High-Performance Distributed Solver for Large-Scale Markov Decision Processes. *Automatic Control Laboratory, arXiv*, 2024. DOI: 10.48550/arXiv.2502.14474

10. Pouhela, F., Krummacker, D., & Schotten, H. Entity Component System Architecture for Scalable, Modular, and Power-Efficient IoT-Brokers. *21st IEEE International Conference on Industrial Informatics (INDIN)*, 2023. DOI: 10.1109/INDIN51400.2023.10218094

11. Papagiannakis, G., Kamarianakis, M., Protopsaltis, A., Angelis, D., & Zikas, P. Project Elements: A computational entity-component-system in a scene-graph pythonic framework, for a neural, geometric computer graphics curriculum. *Eurographics Association*, 2023. DOI: 10.48550/arXiv.2302.07691

12. Jiang, Y., Kang, J., Niyato, D., Ge, X., Xiong, Z., Miao, C., & Shen, X. Reliable Distributed Computing for Metaverse: A Hierarchical Game-Theoretic Approach. *IEEE Transactions on Vehicular Technology*, 2023, vol. 72, pp. 1084–1100. DOI: 10.1109/TVT.2022.3204839

13. Liu, Z., Chu, Y., Li, G., & Zhang, H. A Co-simulation-Based System Using Vico for Marine Operation. *Lecture Notes in Computer Science, Springer International Publishing*, 2023, vol. 13765, pp. 228–241. DOI: 10.1007/978-3-031-26236-4_20

14. Hatledal, L. I., Chu, Y., Styve, A., & Zhang, H. Vico: An entity-component-system based co-simulation framework. *Simulation Modelling Practice and Theory*, 2021, vol. 108, article no. 102243. DOI: 10.1016/j.simpat.2020.102243

15. Redmond, P., Castello, J., Calderón Trilla, J. M., & Kuper, L. Exploring the Theory and Practice of Concurrency in the Entity-Component-System Pattern. *Object-Oriented Programming, Systems, Languages & Applications (OOPSLA), arXiv*, 2025. DOI: 10.48550/arXiv.2508.15264

16. Kholmatova, Z., Siraj, A. H., & Yakovleva, E. Approximating and Predicting Energy Consumption of Portable Devices. *13th International Conference on Networks, Communication and Computing*, 2024, pp. 45–50. DOI: 10.1145/3711650.3711657

17. Almasri, A., El-Kour, T., Silva, L., & Abdulfattah, Y. Evaluating the Energy Efficiency of Popular US Smartphone Health Care Apps: Comparative Analysis Study Toward Sustainable Health and Nutrition Apps Practices. *JMIR Human Factors*, 2024, no. 11, article no. e58311. DOI: 10.2196/58311

18. O'Connor, O., Elfouly, T., & Alouani, A. Survey of Novel Architectures for Energy Efficient High-Performance Mobile Computing Platforms. *Energies*, 2023, vol. 16, article no. 6043. DOI: 10.3390/en16166043

19. Alsharif, M. H., Kelechi, A. H., Abu Jahid, A., Kannadasan, R., Singla, M. K., Gupta, J., & Geem, Z. W. A comprehensive survey of energy-efficient computing to enable sustainable massive IoT networks. *Alexandria Engineering Journal*, 2024, vol. 91, pp. 12–29. DOI: 10.1016/j.aej.2024.01.067

20. Hirsch, M., Mateos, C., & Majchrzak, T. A. Exploring Smartphone-Based Edge AI Inferences Using Real Testbeds. *Sensors*, 2025, vol. 25, article no. 2875. DOI: 10.3390/s25092875

21. Barroso, L. A., & Hölzle, U. The Case for Energy-Proportional Computing. *Computer*, 2007, vol. 40, pp. 33–37. DOI: 10.1109/MC.2007.443

22. Ricciardi, S., Palmieri, F., Fiore, U., Castiglione, A., & Santos-Boada, G. Modeling energy consumption in next-generation wireless access-over-WDM networks with hybrid power sources. *Mathematical and Computer Modelling*, 2013, vol. 58, pp. 1389–1404. DOI: 10.1016/j.mcm.2012.12.004

23. Mahmood, F., Perrins, E., & Liu, L. Energy-Efficient Wireless Communications: From Energy Modeling to Performance Evaluation. *IEEE Transactions on Vehicular Technology*, 2019, vol. 68, pp. 7643–7654. DOI: 10.1109/TVT.2019.2921304

24. Caiazza, C., Luconi, V., & Vecchio, A. Energy consumption of smartphones and IoT devices when using different versions of the HTTP protocol. *Pervasive and Mobile Computing*, 2024, vol. 97, article no. 101871. DOI: 10.1016/j.pmcj.2023.101871

25. Luo, X., Liu, D., Kong, H., Huai, S., Chen, H., Xiong, G., & Liu, W. Efficient Deep Learning Infrastructures for Embedded Computing Systems: A Comprehensive Survey and Future Envision. *ACM Transactions on Embedded Computing Systems*, 2024, vol. 24, no. 21, pp. 1–100. DOI: 10.1145/3701728

26. Mamchych, O., & Volk, M. Estimation of power consumption of mobile devices in cloud computing. *Innovative Technologies and Scientific Solutions for Industries*, 2023, no. 1, pp. 72–82. DOI: 10.30837/ITSSI.2023.23.072

 ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

Received 08.08.2025, Received in revised form 08.04.2026
Accepted date 13.07.2026, Published date 31.07.2026

УНІФІКОВАНА МОДЕЛЬ ТА МЕТОД ПРОГНОЗУВАННЯ ЕНЕРГОСПОЖИВАННЯ В РОЗПОДІЛЕНИХ СИСТЕМАХ ЗІ СТАЦІОНАРНИМИ ТА МОБІЛЬНИМИ ПРИСТРОЯМИ З УРАХУВАННЯМ ТОПОЛОГІЇ

О. О. Мамчич, М. О. Волк

Предметом *цього дослідження є моделювання та прогнозування енергоспоживання у розподілених обчислювальних системах із використанням архітектури «сутність–компонент» (Entity-Component Architecture, ECA).* ***Задача*** *дослідження зосереджена на удосконаленні* ***методу*** *кількісної оцінки та передбачення енергоспоживання розподіленої обчислювальної системи методом симуляції усіх компонентів цієї системи - обчислювальних блоків (CPU, GPU), інтерфейсів передачі даних (LAN, USB-інтерфейси) та мережевої інфраструктури (LAN-комутатори, маршрутизатори) у гетерогенних середовищах.* ***Валідація*** *методу здійснювалася на прикладі стаціонарних обчислювальних мереж (SCN) та мобільних обчислювальних мереж (MCN). У дослідженні особливу увагу приділено вродженим обмеженням традиційних алгебраїчних моделей, натомість обґрунтовується використання модульної, масштабованої та гнучкої ECA-архітектури, що забезпечує енергетичне профілювання шляхом інтеграції прямих та непрямих витрат енергії розподілених обчислювальних елементів.* ***Експериментальна перевірка*** *показала, що запропонована ECA-модель досягає високої точності прогнозування. У конфігураціях SCN симуляції співпали з реальними вимірами енергоспоживання із відхиленням у 4,4% для задач, що виконуються лише на CPU, тоді як завдання CPU+GPU показали більшу розбіжність — близько 20,2% через похибку моделі для недовантаженого GPU. Натомість у сценарії MCN, що охоплював мобільні пристрої та USB-з'єднання, зафіксовано мінімальні відхилення — 3,5% для задач лише на CPU та близько 10,1% для режиму CPU+GPU. Ці результати підтверджують надійність моделі у достатньо точному прогнозуванні енергоспоживання у різних обчислювальних умовах. Ми робимо* ***висновок*** *що ефективний перехід від традиційних алгебраїчних моделей до гнучкої та розширюваної ECA-методології, яка суттєво покращила* ***модульність***, ***масштабованість*** *та* ***універсальність*** *методології. Серед основних досягнень - удосконалення практик обчислювального бенчмаркінгу шляхом* ***узагальнення обчислювальних навантажень****, застосування комплексних вимірювань енергоспоживання та підкреслення енергетичних переваг мобільних платформ порівняно з традиційним стаціонарним обладнанням як для CPU-, так і для GPU-завдань. Дослідження також виявило певні* ***обмеження*** *запропонованого методу, зокрема обмеженість лінійної інтерполяції у моделюванні залежності навантаження від енергоспоживання та нехтування миттєвим станом апаратного забезпечення, зокрема поточною температурою та інерційністю температури, що впливають на миттєву продуктивність і споживання енергії.* ***Наукова новизна*** *даного дослідження складається з двох частин:* ***нового підхода до математичного моделювання****, що дозволяє простіше розширяти та масштабувати модель, та* ***нової більш точної та консистентної моделі*** *розподілених обчислювальних систем*

Ключові слова*: енергоефективність; розподілені обчислення; хмарні обчислення; зелені обчислення; математичне моделювання; смартфон; центральний процесор; графічний процесор; архітектура сутностей-компонентів.*

Мамчич Олександр Олександрович – асп. каф. електронних обчислювальних машин, Харківський національний університет радіоелектроніки, Харків, Україна.

Волк Максим Олександрович – д-р техн. наук, проф. каф. електронних обчислювальних машин, Харківський національний університет радіоелектроніки, Харків, Україна.

Oleksandr Mamchych – PhD Student of the Department of Electronic Computers (ECM), Kharkiv National University of Radioelectronics**,** Kharkiv, Ukraine,
e-mail: mamont0207@gmail.com, ORCID: 0009-0001-6602-2929, Scopus Author ID: 58099399400.

Maksym Volk – Doctor of Technical Science, Professor at the Department of Electronic Computers (ECM), Kharkiv National University of Radioelectronics**,** Kharkiv, Ukraine,
e-mail: maksym.volk@nure.ua, ORCID: 0000-0003-4229-9904, Scopus Author ID: 9636701100.