engineering proceedings

Proceeding Paper

Component-Oriented Adaptation Design Based on Embedded Model–View–Controller Architecture: Virtualization and Cross-Platform Integration of Legacy Devices †

Chen-Liang Wei [1], Hsin Cheng [2] and Hsing-Kuo Peng [3,*]

[1] Department of Electrical Engineering, National Central University, Taoyuan 320317, Taiwan; britewei@gmail.com
[2] College of Management, Yuan Ze University, Taoyuan 320315, Taiwan; tommy08260826@gmail.com
[3] UMiTouch Software Technology Inc., Taichung 403, Taiwan
* Correspondence: hkpeng@gmail.com
† Presented at the 6th International Conference on Electronic Communications, Internet of Things and Big Data (ICEIB 2026), New Taipei, Taiwan, 24–26 April 2026.

Abstract

Integrating legacy devices into IoT ecosystems presents scalability issues due to tightly coupled monolithic designs. To address the issues, we developed a component-oriented adaptation architecture based on an embedded model–view–controller pattern. By decoupling device behavioral models, control logic, and mapping mechanisms, the architecture enables standardized development and modular configuration for heterogeneous hardware. Experimental results showed that the architecture enables communication and control protocols to evolve independently. The design of the architecture effectively reduces engineering complexity in virtualization adaptation, providing a scalable and maintainable path for legacy hardware transformation within unified IoT standards.

Keywords: IoT; embedded MVC pattern; component-oriented; virtualization adaptation

1. Introduction

As technologies in IoT mature, the industry is transforming toward unified interoperability through standards such as Matter. Nevertheless, a substantial proportion of legacy hardware lacks native compatibility with modern Internet Protocol-based ecosystems. Such an integration depends on monolithic firmware architectures in which protocol-specific mapping and low-level hardware control are tightly coupled. The absence of a standardized abstraction layer fosters spaghetti code structures, creating significant barriers to cross-platform migration: any modification to the communication protocol necessitates a complete reconfiguration of the device control logic.

To address these scalability and maintenance challenges, we developed a component-oriented adaptation architecture. In contrast to conventional gateways that perform external protocol translation, the developed architecture restructures internal firmware by adopting an embedded model–view–controller (MVC) pattern. By decoupling the device's behavioral state (model), communication interface mapping (view), and physical execution logic (controller), the architecture establishes a standardized template for virtualizing heterogeneous hardware.

The developed architecture is a modular adaptation framework that isolates hardware drivers from application logic to enhance system maintainability. The architecture ensures a standardized virtualization pathway enabling legacy devices to participate in unified IoT

Check for updates

Academic Editors: Teen-Hang Meen, Shu-Han Liao and Cheng-Fu Yang

Published: 23 November 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

standards such as Matter. The performance evaluation results of the developed architecture showed that the architecture significantly reduces marginal integration costs compared with monolithic designs.

2. Related Work

Smart home systems continue to face significant challenges in interoperability due to heterogeneous devices and fragmented ecosystems. The Matter standard introduces a unified data model and cross-platform control mechanism to enhance interoperability and promote vendor standardization [1,2]. However, its scope is directed toward newly developed devices and does not systematically incorporate legacy hardware lacking connectivity and standardized interfaces [1,2]. To enable cross-platform integration, bridging and interworking mechanisms have been proposed to facilitate interoperability between Matter and energy conservation hierarchical networks lite through protocol translation [3]. Nguyen et al. presented a Matter–one machine-to-machine (oneM2M) proxy to mitigate ecosystem lock-in [4], and further integrated the World Wide Web Consortium (W3C) Web of Things (WoT) using standardized descriptions [5]. These approaches, however, concentrate on protocol and data mapping with limited attention to internal adaptation structures.

Gateway-based solutions are used for the integration of the W3C WoT. For instance, a virtual device-based gateway is proposed to abstract heterogeneous devices and provide unified application programming interfaces (APIs) [6]. However, such designs emphasize centralized management and protocol compatibility without defining modular firmware structures for encapsulating legacy devices [6]. From a software perspective, component-based architectures are recognized for enhancing modularity and scalability in IoT systems [7]. Research on embedded systems underscores the importance of clear architectural separation under resource constraints [8]. The MVC layers, which separate data (model), control logic (controller), and interfaces (view), have been widely adopted in software engineering [9,10], but their application to embedded device adaptation remains limited. Meanwhile, implementation-oriented studies demonstrate practical Matter deployment on embedded platforms [11], though these efforts primarily focus on native device development rather than legacy adaptation.

Overall, existing studies address interoperability, protocol translation, virtualization, and modular design separately, lacking a unified approach that integrates legacy device virtualization with structured embedded architecture. Therefore, a component-oriented embedded adaptation architecture with MVC-based separation is required to enable the scalable and maintainable integration of legacy devices. Table 1 compares the existing approaches with the proposed framework in terms of research focus, legacy device support, device virtualization, platform mapping, and software architecture design.

Table 1. Comparison of existing approaches and the developed architecture.

Feature	Bridge	Interworking	Gateway	Component IoT	Proposed Framework
Research focus	Protocol bridging	Cross-platform interoperability	Multi-device integration	Modular IoT architecture	Embedded adaptation architecture
Legacy device supports	Partial	Limited	Moderate	Not addressed	Fully supported
Device virtualization	Low	Low	Medium	Medium	High
Platform mapping	Protocol-level	Data model mapping	API abstraction	Not explicit	Explicit mapping layer (View)

Table 1. *Cont.*

Feature	Bridge	Interworking	Gateway	Component IoT	Proposed Framework
Software architecture design	Weak	Weak	Partial	Strong	Strong (MVC-based)
Decoupling level	Low	Medium	Medium	High	High (Model–View–Controller separation)
Component modularity	Low	Low	Medium	High	High (component-oriented design)
Embedded system focus	Limited	Limited	Moderate	Not focused	Designed for microcontrollers
Scalability	Limited	Moderate	Moderate	High	High (linear integration cost)
Reusability	Low	Low	Medium	High	High
Architecture methodology	Not defined	Not defined	Partial	Defined	Clearly defined (MVC + components)

3. System Design

3.1. Architecture

We developed a component-oriented embedded endpoint architecture that integrates legacy devices into the Matter ecosystem. The system was implemented using a Waveshare ESP32-C6-DEV-KIT-N8 development board (Waveshare, Shenzhen, China), equipped with an Espressif ESP32-C6 microcontroller. The firmware was developed using ESP-IDF version 5.5.2 and ESP-Matter version 1.4.2, incorporating FreeRTOS version 10.5.1 (Espressif-modified version) and the Espressif-maintained OpenThread branch (commit 36b14d3ef74f) bundled with ESP-IDF version 5.5.2. The architecture comprises three layers (Figure 1). The architecture includes MatterCore and EndpointRegistry, responsible for Matter initialization, event handling, and routing of attributes and commands. The device layer encapsulates devices such as relays, fans, and sensors as independent layers registered to the core. The hardware interface layer connects to peripherals through the ESP-IoT development framework drivers, including general-purpose input/output, pulse-width modulation, and analog-to-digital converter. This separation enables modular integration without modifying the system core.

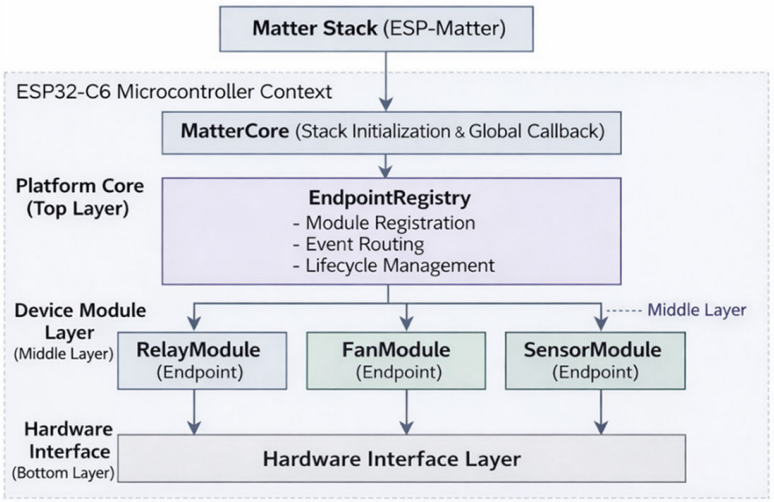

Figure 1. The developed architecture of the component-oriented embedded endpoint platform.

3.2. Embedded MVC Module

To achieve low coupling and consistent adaptation, each device layer adopts an embedded MVC structure. This design is applied internally within each layer to separate device behavior, control logic, and platform mapping (Figure 2). The developed architecture maintains device states as the single source of truth. The controller processes control semantics, executes device logic, and interacts with hardware drivers. The view maps device states to Matter attributes and handles synchronization with the platform. Hardware drivers directly interface with physical components and remain independent from upper-layer logic. This design ensures clear separation between state management, control execution, protocol mapping, and hardware interaction, improving modularity and reusability.

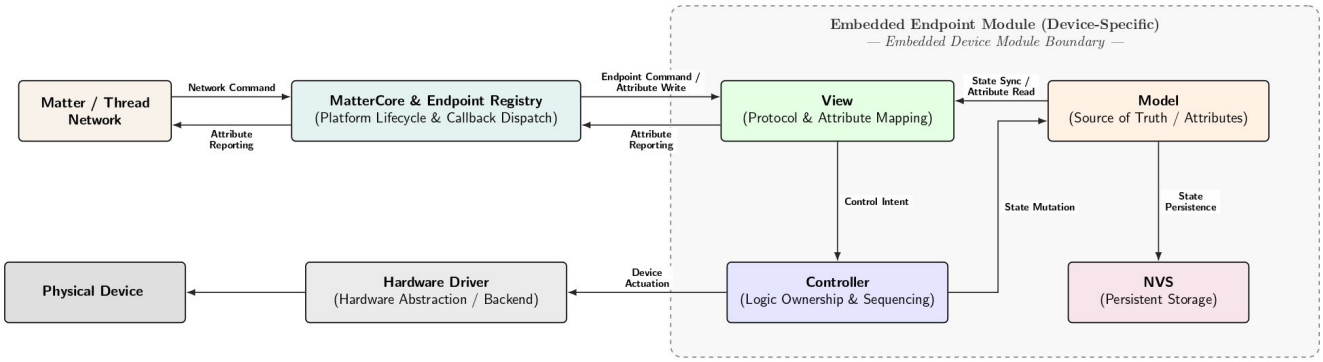

Figure 2. MVC architecture of the embedded endpoint layer.

3.3. Virtualization and Endpoint Mapping

In the architecture, legacy devices are integrated by encapsulating them as independent layers and registering them with the EndpointRegistry. Each layer corresponds to a virtualized endpoint defined within the Matter data model. In operation, device inputs are processed by the controller layer and transformed into internal state representations, which are maintained in the model layer. The view layer maps these states to corresponding Matter attributes and exposes them through standardized endpoints. Through this mechanism, legacy devices without native compatibility are represented as virtualized entities within the system, enabling consistent interaction with external platforms while preserving original device behavior.

3.4. System Operation

The architecture operates through a structured interaction between the Matter stack and device layers. In the control flow, commands received by the platform core are routed through the EndpointRegistry to the corresponding layer. The controller layer interprets the command, executes device-specific logic, and updates the model layer. The updated state is then synchronized to the platform through the View. In the reverse flow, device state changes are processed and updated in the model layer and propagated through the view layer. This interaction ensures that command execution and state synchronization follow a consistent and modular process.

3.5. Architectural Formalization of Embedded MVC Adaptation

To generalize the developed architecture beyond implementation-specific descriptions, the embedded MVC-based adaptation architecture is formulated as a reusable structural model for legacy device virtualization. Let a device layer be defined as a quadruple:

$$M_d = \{M, C, V, D\} \quad (1)$$

where M denotes the state representation (model), C is the control logic (controller), V is the platform mapping layer (view), and D is the hardware driver (physical interface abstraction). In this structure, the data and control flow can be formally described as:

$$Input \rightarrow C \rightarrow M \rightarrow V \rightarrow Platform \quad (2)$$

and for reverse synchronization:

$$Device\ State\ Change \rightarrow D \rightarrow C \rightarrow M \rightarrow V \rightarrow Platform \quad (3)$$

The equations highlight that all external interactions are mediated through the Controller, while the Model serves as the single source of truth. The view layer is responsible for mapping internal states to external platform representations, ensuring that protocol-specific logic is isolated from device behavior. The architecture enforces the following decoupling constraints:

- $C \perp V$: Control logic is independent of platform mapping;
- M is the sole authoritative state holder;
- $D \perp Platform$: Hardware interaction is isolated from protocol semantics;
- Core platform components do not contain device-specific logic.

Under these constraints, the integration of a new device layer does not require modification of existing layers or system core logic, but only the registration of the model layer into the platform registry. This structure transforms the integration problem from a code modification process into a component composition process. Consequently, the marginal integration cost becomes bounded and predictable, enabling scalable system evolution.

4. Implementation and Evaluation

The developed architecture was implemented using the ESP32-C6 microcontroller, which natively supports Wireless Fidelity (Wi-Fi) 6 and Matter over Thread. The firmware was developed on FreeRTOS using the ESP-Matter SDK and OpenThread stack. To evaluate the architectural efficiency, we compared the developed MVC architecture with a Monolithic Baseline. Three metrics were used as proxies for coupling and complexity: lines of code (LoC), modified file count, and shared-core modification count to ensure experimental consistency. AI-assisted coding agents (Codex GPT-5.4) were utilized for implementation following strict architectural constraints.

The first experiment was conducted to assess the entry cost of integrating a single relay endpoint. As shown in Figure 3, the monolithic baseline required significantly less initial effort, involving only 318 LoC and 3 modified files. In contrast, the developed architecture required 631 LoC and 16 files for the same task [Figure 3a,b]. The shared-core modification count [Figure 3c], was 4 compared with 0 for the baseline.

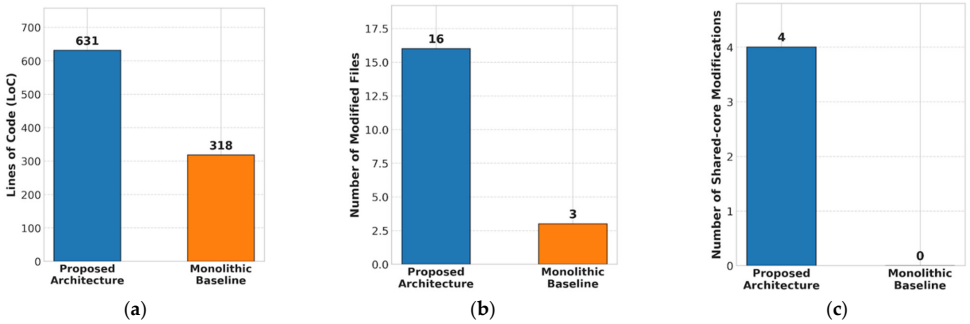

Figure 3. Initial integration cost evaluation for a single relay endpoint: (**a**) LoC; (**b**) number of modified files; (**c**) number of shared-core modifications.

This result shows a structural fixed overhead. The MVC-based approach demands a higher initial investment to establish modular boundaries and standardized interfaces. However, such an investment is a deliberate trade-off to ensure long-term structural clarity and prevent coupling proliferation.

4.1. Continuous Scaling and Marginal Cost Evaluation

The second experiment was conducted to evaluate the evolutionary resilience of the developed architecture by sequentially adding three heterogeneous endpoints: a pump relay, a vent fan, and a temperature sensor. The results show that the architectural complexity remains strictly controlled, although functional code (LoC) increases with new features. The shared-core modification counts remained constant at four files across all three expansion rounds (Figure 4). The developed architecture presents core stability; adding new devices does not require invasive changes to the system's foundation. This effectively solves the coupling explosion typically seen in monolithic designs.

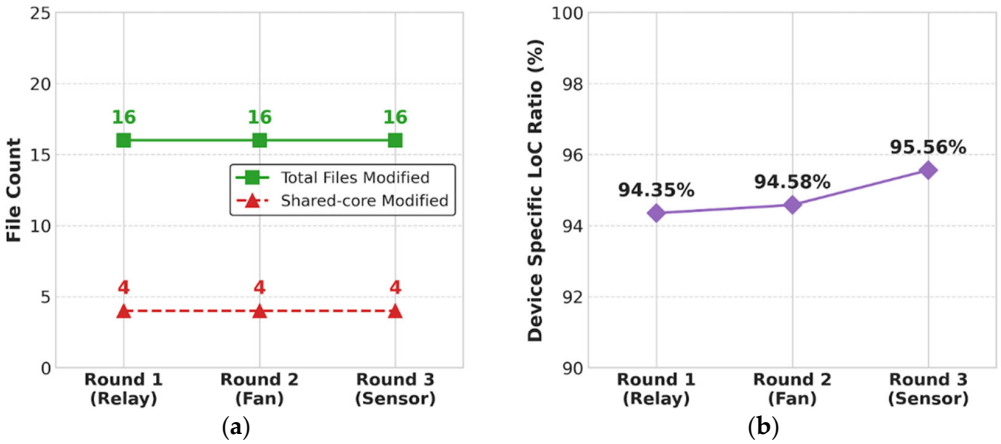

Figure 4. Marginal cost and stability trends during continuous expansion of three heterogeneous endpoints: (**a**) total files modified and shared-core modified; (**b**) Device-specific LoC ratio (%).

Figure 4b shows that a code isolation rate consistently exceeds 94% (94.35% to 95.56%). This high isolation rate suggests that the MVC structure successfully confines device-specific logic within modular boundaries. By separating state management (model) from protocol mapping (view), the architecture ensures that the integration effort is localized. These results validate that the developed architecture enables a highly scalable and maintainable path for legacy hardware transformation, keeping the marginal engineering cost predictable.

4.2. Discussion

The monolithic baseline provides a lower entry barrier but accumulates complexity as the system evolves. The developed architecture introduces a fixed structural overhead to establish modular boundaries, resulting in an initial LoC count approximately twice that of the baseline. This overhead, however, is negligible relative to the 4- and 8-megabyte flash capacities of the ESP32-C6 microcontroller, and the trade-off is justified by avoiding spaghetti code architectures. In contrast to monolithic designs, the architecture maintains bounded coupling, with shared-core modifications limited to four files regardless of system scale. More than 94% of code changes are confined to independent layers, ensuring that existing endpoints remain unaffected when new hardware is introduced. This high degree of isolation validates the architecture as a scalable and maintainable pathway for transforming legacy hardware within unified IoT standards.

4.3. Generalization Beyond Matter Platforms

Although the developed architecture is implemented using the Matter protocol and the ESP32-C6 platform, the design principles are not limited to a specific communication standard or hardware environment. In the architecture, the view component functions as an abstraction layer for platform-specific mapping. This design enables the same device layer structure to be adapted to different IoT ecosystems by replacing or reconfiguring the view layer, without modifying the model or controller. The architecture can be generalized to Message Queuing Telemetry Transport-based IoT systems, where the view layer maps device states to topic-based messaging, representational state transfer or hypertext transfer protocol-based systems, where the view exposes device states through APIs, WoT, in which the view layer generates standardized thing descriptions, and industrial IoT platforms, where device states are mapped to fieldbus or OPC-UA representations. This separation ensures that protocol evolution or platform migration does not affect internal device logic, significantly reducing system refactoring costs. Therefore, the embedded MVC adaptation architecture is interpreted not as a Matter-specific solution, but as a generalizable design pattern for heterogeneous device virtualization and cross-platform integration.

4.4. Validity

Despite the promising results, several limitations must be noted. First, the implementation relies on AI-assisted coding to standardize development, which may introduce bias in metrics such as LoC and file structure. Second, the evaluation is conducted on a limited set of device types, including relays, fans, and sensors, which do not represent complex or stateful devices. Third, the experiments are performed on a specific embedded platform (ESP32-C6 with ESP-Matter), and variations in hardware or system environments may affect generalizability. This study focuses on software engineering metrics rather than runtime performance. Therefore, conclusions regarding execution latency, memory consumption, and computational efficiency are beyond the scope of this evaluation.

5. Conclusions

A component-oriented adaptation architecture was developed based on embedded MVC layers to address scalability and coupling challenges in legacy device integration. The evaluation results of the architecture show that, although the framework incurs a fixed structural overhead during initial implementation (631 versus 318 lines of code), it effectively mitigates complexity proliferation as the system evolves. By isolating device behavior, control logic, and platform mapping, the architecture achieves a high code isolation rate (94.35–95.56%), maintaining shared-core modifications at approximately four files. Whereas conventional approaches treat legacy integration as a protocol translation task, the developed architecture reframes the problem as one of architectural design, where device integration is realized through modular composition rather than invasive modification. Using this perspective, predictable integration behavior and continuous system expansion can be achieved without compromising stability. The architecture is not confined to Matter-based implementations. By conceptualizing the view layer as a platform mapping layer, the same modular structure can be extended to other IoT ecosystems with minimal adaptation. In practice, each device can be systematically encapsulated as an independent MVC-based layer with isolated state management and mapping mechanisms.

Author Contributions: Conceptualization, H.-K.P.; methodology, H.-K.P.; software, C.-L.W.; validation, C.-L.W.; formal analysis, C.-L.W.; investigation, H.C.; data curation, H.C.; writing—original draft preparation, H.C.; supervision, H.-K.P. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data supporting the findings of this study are presented in the article. The underlying experimental datasets, testing records, and source code are not publicly available due to intellectual property and proprietary information restrictions of UMiTouch Software Technology Inc.

Conflicts of Interest: Author Hsing-Kuo Peng was employed by UMiTouch Software Technology Inc. and serves as the company's representative. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

References

1. Belli, D.; Barsocchi, P.; Palumbo, F. Connectivity Standards Alliance Matter: State of the art and opportunities. *Internet Things* **2024**, *25*, 101005. [CrossRef]
2. Madadi-Barough, S.; Ruiz-Blanco, P.; Lin, J.; Vidal, R.; Gomez, C. Matter: IoT Interoperability for Smart Homes. *IEEE Commun. Mag.* **2025**, *63*, 106–112. [CrossRef]
3. Pham, V.C.; Nguyen-Mau, T.; Sioutis, M.; Tan, Y. Matter and ECHONET Lite: Similarities, differences, and a bridge solution for interoperability. *Internet Things* **2024**, *27*, 101265. [CrossRef]
4. Thi, D.L.N.; Kieu, X.T.; Bui, T.S.; Le, T.L.; Pham, V.C. Towards interworking of matter and oneM2M: Design and implementation of a matter–oneM2M Interworking Proxy Entity. *Internet Things* **2024**, *27*, 101313. [CrossRef]
5. Nguyen, H.-T.; Pham, V.-H.; Vu, V.-T.; Nguyen, T.-T. Web access as a service for CSA Matter protocol: Bridging Matter and the W3C Web of Things framework for smart home interoperability. *Internet Things* **2025**, *32*, 101618. [CrossRef]
6. Phan, L.A.; Kim, T. Breaking Down the Compatibility Problem in Smart Homes: A Dy-namically Updatable Gateway Platform. *Sensors* **2020**, *20*, 2783. [CrossRef] [PubMed]
7. Lo, S.K.; Liew, C.S.; Tey, K.S.; Mekhilef, S. An Interoperable Component-Based Architecture for Data-Driven IoT System. *Sensors* **2019**, *19*, 4354. [CrossRef] [PubMed]
8. Mendoza, J.F.; Ordóñez, H.; Ordóñez, A.; Jurado, J.L. Architecture for embedded software in microcontrollers for Internet of Things (IoT) in fog water collection. *Procedia Comput. Sci.* **2017**, *109*, 1092–1097. [CrossRef]
9. Buschmann, F.; Meunier, R.; Rohnert, H.; Sommerlad, P.; Stal, M. *Pattern-Oriented Software Architecture: A System of Patterns*; John Wiley and Sons: Hoboken, NJ, USA, 2008; Volume 1.
10. Krasner, G.E.; Pope, S.T. A cookbook for using the model-view controller user interface paradigm in Smalltalk-80. *J. Object-Oriented Program.* **1988**, *1*, 26–49.
11. Mota, A.; Serôdio, C.; Valente, A. Matter Protocol Integration Using Espressif's Solutions to Achieve Smart Home Interoperability. *Electronics* **2024**, *13*, 2217. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.