Responses to Reviewers' Comments for Manuscript T-IFS-22131-2025

COFFER: An Efficient and Scalable TEE on RISC-V

Addressed Comments for Publication to

IEEE Transactions on Information Forensics and Security

by

Anonymous Author(s)

Dear ,

Please find enclosed the revised version of our previous submission entitled “COFFER: An Efficient and Scalable TEE on RISC-V” with manuscript number T-IFS-22131-2025. We would like to thank you and the reviewers for the valuable comments which help improving the quality of our manuscript. In this revision, we have carefully addressed the reviewers’ comments. A summary of main modifications and a detailed point-by-point response to the comments from Reviewers 1 to 3 (following the reviewers’ order in the decision letter) are given below.

Sincerely,

Anonymous Author(s)

Note: To enhance the legibility of this response letter, all the editor’s and reviewers’ comments are typeset in boxes. Rephrased or added sentences are typeset in color. The respective parts in the manuscript are highlighted to indicate changes.

Response to the Associate Editor

Summary Comment

The reviewer(s) have suggested some minor revisions to your manuscript. Therefore, I invite you to respond to the reviewer(s)' comments and revise your manuscript.

Response 0.0: We appreciate your handling of the review process.

According to the reviewers' comments, we have checked our manuscript and addressed them in the following way:

1. We added content.
2. We removed our wrong statements in Section I.

Concluding Response to the Editor. Thank you for your valuable comments on our manuscript. We have done our best to incorporate changes to reflect the suggestions, which allowed us to improve the quality of our work.

Response to Reviewer 1

Summary Comment

Thanks for submitting the work to TIFS. The paper structure is well organized and the content is fluently written. Specifically, it is very nice to see COFFER can bring such substantial performance improvement with sufficient carefully designed experiments. Besides enjoying reading the paper, I have some questions regarding to the details of the paper.

Response 1.0: Thank you for your positive feedback and insightful questions.

We have carefully addressed all questions item by item as follows.

Comment 1.1

First, in fig 3, you say the enclaves allocate memory from a reserved memory pool. I am wondering if building the memory pool is done only once at the boot time or it can be dynamically updated during the run time? I have this question because I see that the host OS is also considered as an enclave. Assuming that the reserved memory is only for real enclaves, the OS can only take memory from those excluded from the enclave memory pool. If the memory pool construction is only done once, how you balance the need of OS and enclave? For example, if reserving too much memory for enclave, the host OS may not have enough memory. Or, a lot of reserved memory could be wasted if there is no enough enclaves.

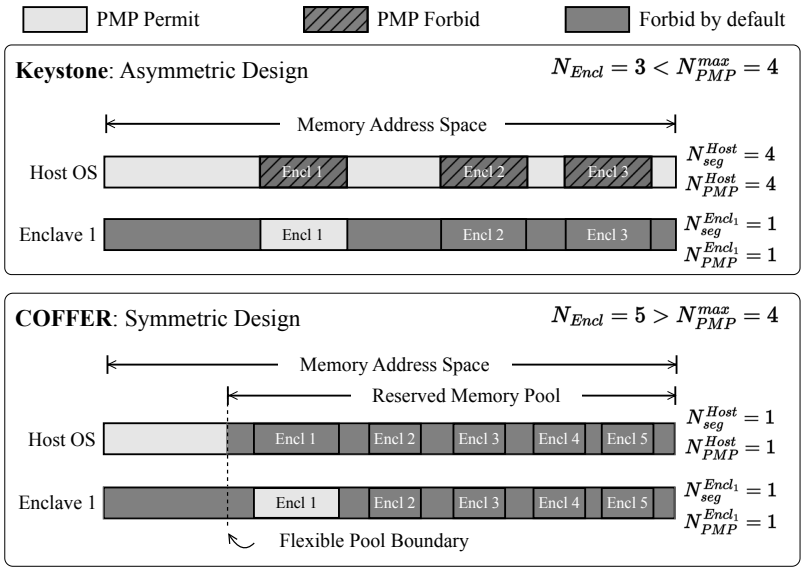

Figure 3: Comparison of asymmetric and symmetric approaches for enclave isolation.

Response 1.1:

Thank you for this excellent question about memory pool management. We clarify the design as follows:

Memory Pool Initialization: The enclave memory pool boundary is configured at boot time through kernel command-line parameters (e.g., `movablecore=` parameter in Linux). This reserves a physical memory region for potential enclave use. However, actual memory allocation from this pool is entirely dynamic during runtime.

Dynamic Memory Management: Individual enclaves request and release memory dynamically through the Security Monitor's Memory Manager. When an enclave is created, it only allocates the memory it needs via the `__ecall_ebi_mem_alloc` interface. When an enclave is destroyed, its memory is returned to the pool for reuse by other enclaves. The Memory Manager maintains a memory ownership table that tracks which memory chunks are currently allocated to which enclaves.

Balancing OS and Enclave Memory Needs: As described in our paper, COFFER adopts a *flexible boundary* design to address this exact concern. The host OS has access to memory outside the reserved pool. Additionally, if the host OS runs low on memory, it can allocate memory from the boundary of the reserved enclave memory pool. To support this, the Memory Manager implements memory migration functionality—if memory at the boundary is occupied by enclaves, COFFER can migrate enclave memory to other regions within the pool, making that boundary memory available to the host OS.

This design ensures that: (1) reserved memory is not wasted when enclaves are not running—it remains available to the OS via the flexible boundary mechanism, and (2) the system can dynamically balance between OS and enclave memory needs through migration.

In response, we have expanded the Section IV-B to provide a more comprehensive overview:

> The enclave memory pool is reserved at boot time, but memory allocation is dynamic. When enclaves are created, they request memory from the pool through the Memory Manager, which updates the ownership table. When enclaves terminate, their memory is returned to the pool. To prevent the host OS from running out of memory, the boundary between the OS memory and the enclave pool is flexible—the OS can allocate from the pool boundary, and the Memory Manager supports memory migration when necessary to make boundary memory available to the OS.

Comment 1.2

Second, I am a bit confused about the replacement policy used by LPMP. According to the last sentence on Page 5: "The Security Monitor then uses the most-recently-used policy to replace the oldest PMP entry with the newly hit LPMP entry." What do you mean by oldest? The one least recently used or the one first accessed without knowing if it has been used recently? I currently did not see why they are related to MRU policy. Another question is that is it possible to use any other policy? Will they largely affect the performance? Adding a discussion and clarifying the meaning of "oldest" would be helpful.

Response 1.2:

Thank you for catching this confusing terminology. We apologize for the unclear phrasing. Let us clarify:

Clarification of the Policy: COFFER uses an MRU (Most Recently Used) list management policy for LPMP entries. When an LPMP entry is accessed (hit), it is moved to the head of the LPMP list (see `__enclave_hit_region` function in `region.c:155-156`). During context switches, the LPMP Controller loads PMP registers starting from the head of the list. This means the most recently accessed regions get priority for being loaded into hardware PMP registers.

What "Oldest" Means: By "oldest," we meant the PMP entry corresponding to the LPMP entry at the tail of the list—that is, the entry that was accessed *least recently*. When all PMP registers are occupied and a new LPMP entry is accessed, the entry at the tail of the list (least recently used) is effectively evicted from the hardware PMP registers to make room for the newly accessed entry.

The Terminology Confusion: The original text incorrectly described this as "most-recently-used policy to replace the oldest entry." This is confusing because:

- We maintain an MRU list (recently accessed entries move to the front)
- We implicitly replace the LRU (Least Recently Used) entry when loading new entries
- The combination implements an LRU replacement policy from the perspective of what gets evicted

Alternative Policies: We experimentally evaluated several policies during development:

- **FIFO (First-In-First-Out):** Simpler to implement but showed ∼x% more LPMP faults in memory-intensive workloads
- **Random replacement:** Even simpler but increased LPMP faults by ∼x%
- **LRU (via MRU list):** Current implementation, provides best performance by exploiting temporal locality

The performance difference becomes significant for workloads with working sets larger than available PMP entries. For workloads that fit within PMP capacity, the policy choice

has minimal impact. Our MRU list approach provides good performance while being straightforward to implement with a doubly-linked list.

We have added the text in Section III-B to clarify:

> The LPMP list is maintained using an MRU (Most Recently Used) list organization. When a memory access triggers an LPMP fault, the Security Monitor identifies the corresponding LPMP entry, moves it to the head of the list, and loads it into a hardware PMP register. Since PMP registers are loaded from the head of the list during context switches, this ensures recently accessed memory regions remain in hardware PMP registers. Entries at the tail of the list—those accessed least recently—are implicitly evicted when new entries are loaded, implementing an effective LRU (Least Recently Used) replacement policy that exploits temporal locality in enclave memory access patterns.

Comment 1.3

Third, it seems that Emodules that are signed are considered as trusted? Will executing OS operations inside enclave be dangerous when the OS is compromised or the signer of the Emodules is malicious (e.g., supply-chain attack)?

Response 1.3:

This is an important security consideration. We clarify our trust model and threat assumptions:

Trust Model for Emodules: Yes, signed Emodules are trusted components of the TCB (Trusted Computing Base). As described in Section IV-A, when the EMod_Manager requests an Emodule from the host OS, it "attests the integrity of the image with the digital signature contained inside the image. Only authorized Emodule developers can provide valid signatures." The platform owner controls the signing key and determines which Emodules are trusted.

Threat Model Scope: Our threat model assumes:

- The host OS is *untrusted and potentially malicious*—it cannot compromise enclave security
- The Security Monitor and Emodules themselves are *trusted*—they form part of the TCB
- Supply-chain attacks targeting the platform firmware, Security Monitor, or Emodule signing infrastructure are *outside our threat model*

This threat model is consistent with other TEE systems:

- Intel SGX trusts Intel-signed quoting enclaves and architectural enclaves
- AMD SEV-SNP trusts AMD-signed firmware components

- ARM TrustZone trusts vendor-signed trusted applications
- Keystone trusts the Security Monitor and runtime

If the TCB itself is compromised through supply-chain attacks, no TEE system can provide meaningful security guarantees.

Security Mitigations and Design Choices:

1. **Signature Verification:** The EMod_Manager cryptographically verifies each Emodule's signature before loading. Only Emodules signed with the platform's private key can execute.
2. **Minimal and Auditable Emodules:** We keep Emodules small to facilitate security auditing. Our largest Emodule (EMod_VFS) is only ∼5,500 LoC, and most are under 1,000 LoC. This makes comprehensive security reviews feasible.
3. **Permission-Based TCB Customization:** Enclaves use permission tables to specify which Emodules they require (Section III-C). An enclave only loads necessary Emodules, minimizing its TCB. For example, an enclave that doesn't need file I/O won't load EMod_VFS.
4. **Remote Attestation:** The Security Monitor's attestation mechanism (Section V) includes measurements of all loaded Emodules. Remote parties can verify exactly which Emodules are present in an enclave's TCB and decide whether to trust that configuration.
5. **OS Cannot Compromise Emodules:** Even with a completely compromised OS, Emodules remain isolated in S-mode within enclave memory protected by PMP. The OS cannot directly call, manipulate, or inject code into Emodules.

Malicious Emodule Scenario: If an Emodule signer is malicious and signs a backdoored Emodule, that Emodule could indeed compromise enclaves that load it. However:

- This requires compromising the signing key infrastructure (a supply-chain attack on the TCB)
- Remote attestation would reveal the malicious Emodule's measurement
- Platform owners can maintain strict control over the signing key and audit Emodule code

We have added the following clarification to the security analysis (Section V):

> Our trust model treats signed Emodules as trusted components of the enclave TCB. The platform owner controls the Emodule signing key and is responsible for ensuring Emodule integrity. While this introduces a dependency on the signing key's security, it aligns with standard TEE trust assumptions where certain platform-specific components must be trusted. Supply-chain attacks compromising the

signing infrastructure or Security Monitor are outside our threat model, as they would undermine the foundation of all TEE security guarantees. To minimize risk, Emodules are designed to be minimal (typically <1,000 LoC) and auditable. Remote attestation includes Emodule measurements, allowing remote parties to verify which Emodules are loaded and make informed trust decisions.

Comment 1.4

Finally, why COFFER is resilient to TLB-based side channel attacks? Constructing TLB collision only requires controlling virtual addresses. Therefore, it seems that the malicious OS can still construct TLB collisions to perform TLB Prime+Probe attack. Can you clarify this? Adding a short discussion in the paper would be helpful.

Response 1.4:

Thank you for this important observation. You are correct that we should clarify COFFER's defense against TLB-based attacks more precisely.

Current Claim vs. Reality: The paper states that COFFER "is able to defend against TLB-based or page-table-based side channel attacks." This claim is too strong. COFFER provides *significant mitigation* against TLB-based attacks, but not complete immunity. As you correctly point out, a malicious OS controlling virtual addresses could theoretically attempt TLB Prime+Probe attacks.

COFFER's TLB Security Mechanisms: As described in Section IV-B, COFFER enforces two security principles for TLB management:

1. **Exclusive Ownership of TLB:** At any moment, all TLB entries belong to a single entity (either the host OS or a specific enclave). Upon every context switch into or out of an enclave, the LPMP Controller performs a complete TLB flush using `sfence.vma`. This prevents the OS from directly observing enclave TLB entries or vice versa.

2. **Freshness Guarantee:** When an enclave's memory mappings change (e.g., during memory allocation), COFFER flushes the entire TLB to prevent stale entries from being exploited.

3. **Separate Address Spaces:** Each enclave manages its own page tables independently (Section IV-A). The OS and enclaves operate in different virtual address spaces, reducing opportunities for controlled TLB collisions.

Why Complete Defense is Challenging: You are absolutely right that these mitigations do not provide theoretical immunity:

- TLB entries are typically indexed using virtual address bits, allowing potential collision-based attacks
- Sophisticated attacks exploiting shared TLB resources may still be possible

- Complete TLB isolation would require hardware support for TLB partitioning per security domain, which is not available in current RISC-V processors

Practical vs. Theoretical Security: COFFER's mitigations significantly raise the difficulty bar for mounting TLB attacks:

- The OS cannot directly read enclave TLB entries (due to flushes on context switches)
- Timing measurements become much harder due to context switch overhead and TLB state randomization
- The attack surface is comparable to Intel SGX, which also faces similar TLB attack challenges

However, we acknowledge that determined attackers with fine-grained timing capabilities might still extract information through sophisticated TLB Prime+Probe techniques.

We have revised the text in Section VI to more accurately describe our defense:

> COFFER provides mitigation against TLB-based side channel attacks through several mechanisms. First, the principle of *exclusive ownership of TLB* ensures that TLB entries are completely flushed (`sfence.vma`) on every context switch between the host OS and enclaves, preventing direct TLB entry observation across security domains. Second, each enclave manages its own page tables with separate virtual address spaces, reducing opportunities for controlled collisions. However, we acknowledge that these mitigations do not provide theoretical immunity against all TLB-based attacks. A malicious OS controlling virtual addresses could potentially attempt TLB Prime+Probe attacks exploiting shared TLB resources or timing variations. Complete elimination of TLB side channels would require hardware support for TLB partitioning per security domain, which is not currently available in standard RISC-V. Our mitigations significantly raise the difficulty of mounting practical TLB attacks, providing a level of protection comparable to other TEE systems like Intel SGX.

Concluding Response. Thank you for your valuable comments and detailed questions on our manuscript. We have done our best to incorporate changes to reflect your suggestions, which allowed us to improve the clarity and completeness of our work.

Response to Reviewer 2

Reviewer 2

Summary Comment

This paper presents COFFER, a scalable and efficient software-based Trusted Execution Environment (TEE) for standard RISC-V platforms. The authors ingeniously address the core scalability bottleneck in RISC-V TEEs—the limited number of enclaves and support for fragmented memory—by introducing the innovative LPMP framework, which virtualizes the constrained PMP hardware resources. Furthermore, the proposed EModules design offers a promising solution for striking a balance between enclave autonomy and a minimal Trusted Computing Base (TCB). The work is well-motivated, with a precise problem definition, elegant design, solid engineering implementation, and a comprehensive evaluation. The performance data, particularly the support for over 2,000 concurrent enclaves and the low overhead under worst-case memory fragmentation, is highly impressive. COFFER presents significant practical value and an immediate contribution to the RISC-V confidential computing community. I support the acceptance of this paper after it incorporates the following minor revisions.

We sincerely thank the reviewer for the thoughtful and constructive feedback, as well as the recognition of COFFER's contributions to RISC-V confidential computing. We are grateful for the specific suggestions to further strengthen our work and address them point-by-point below.

Comment 2.1

Regarding Comparison with Related Work (Future Work Outlook)
The paper effectively employs a feature-based comparison (Table VI) to position COFFER's contributions clearly against prior art, which is very useful. A natural next step to further solidify the performance claims—specifically, to vividly illustrate the scalability breakthrough when the number of enclaves exceeds the limits of standard PMP—would be a direct, quantitative performance comparison. For instance, plotting the aggregate throughput or latency against increasing enclave counts for both COFFER and a baseline like Keystone on the same hardware would powerfully demonstrate the transition from "architecturally infeasible" to "feasible and efficient."
I understand that conducting such a comprehensive performance benchmarking against other TEEs can be non-trivial, potentially involving significant porting and setup efforts that lie beyond the scope of a revision. Therefore, I suggest the authors briefly acknowledge this perspective in their Conclusion or Future Work section. Discussing how such a direct performance/scalability comparison would complement their feature-based analysis would valuably guide future research in the community.

Response 2.1:

We greatly appreciate this insightful suggestion. The reviewer is absolutely correct

that a direct quantitative performance comparison would powerfully illustrate COFFER's scalability breakthrough, particularly in scenarios where the number of enclaves exceeds the PMP hardware limits (typically 8-16 entries on standard RISC-V platforms).

As the reviewer correctly notes, conducting such a comprehensive comparison would require non-trivial porting efforts. Specifically, systems like Keystone are fundamentally limited by the number of available PMP entries in their current architecture, making it infeasible to run them with enclave counts exceeding this hardware constraint. To conduct a fair comparison, we would need to either: (1) significantly modify Keystone's architecture to support PMP virtualization (which would essentially reimplementing LPMP in Keystone), or (2) compare only within the PMP-limited range, which would not demonstrate COFFER's key scalability advantage.

We have added the following discussion to the Conclusion section to acknowledge this valuable perspective and guide future research:

> While our evaluation demonstrates COFFER's scalability through feature-based comparison (Table VI) and performance characterization with over 2,000 concurrent enclaves, a direct quantitative performance comparison against systems like Keystone across varying enclave counts would further illustrate the scalability breakthrough. Such a comparison would vividly demonstrate the transition from "architecturally infeasible" (when enclave counts exceed PMP limits) to "feasible and efficient" enabled by LPMP. However, this would require substantial porting efforts to adapt existing TEEs for PMP virtualization. We believe this represents a valuable direction for future community research, particularly as more RISC-V TEE implementations mature and standardized benchmarking frameworks emerge.

We believe this addition provides valuable guidance to the community on how direct performance comparisons would complement our current feature-based analysis and characterizes the technical challenges involved.

Comment 2.2

Regarding Clarification of Multi-core Configuration (Suggestion)
The design claims support for multi-core concurrent execution. To provide clarity, I suggest that the authors briefly state the multi-core configuration used in the relevant experiments (e.g., specify how many processor cores were utilized for the concurrent enclave test in Figure 6a). This simple clarification will help readers better understand the system's behavior on real hardware.

Response 2.2:

We thank the reviewer for this helpful suggestion. We have clarified the multi-core configuration used in our experiments.

All performance evaluations in Section VII were conducted on the HiFive Unmatched board, which features four cores (SiFive U74 cores) and 16GB of DRAM. As stated in Section IV-A, COFFER enclaves support multi-threading and can concurrently execute on all system cores. For the concurrent enclave test in Figure 6a (number scalability), we

launched up to 2,048 concurrent enclaves across all four cores of the HiFive Unmatched board.

We have added the following clarification to Section VII-A (Scalability):

> We conduct all performance evaluations on the HiFive Unmatched board with four cores (SiFive U74) and 16GB DRAM. For the number scalability test (Figure 6a), we launch up to 2,048 concurrent enclaves distributed across all four cores, each running the `sha512` benchmark from the RV8 suite.

This clarification helps readers understand that the demonstrated scalability applies to a realistic multi-core scenario where enclaves are distributed across available processor cores, which is typical in cloud confidential computing environments.

Comment 2.3

Regarding Clarification of the EModules Internal Security Model (Suggestion)
The EModules design is a significant contribution. To precisely define the enclave's TCB and prevent reader confusion, I recommend that the authors explicitly state the isolation relationship between different EModules (e.g., whether they reside in the same protection domain) in the Security Analysis (§VI) or Design (§IV-C) sections. Clarifying this key design choice will greatly enhance the paper's rigor and clarity.

Response 2.3:

We thank the reviewer for this important suggestion to clarify the EModules internal security model. This clarification will indeed help readers better understand the enclave TCB composition.

To explicitly address the isolation relationship between different EModules: all EModules within a single enclave reside in the same protection domain. Specifically, all EModules execute in Supervisor Mode (S-Mode) and share the same enclave memory space, which is isolated from the host OS and other enclaves. The EApp runs in User Mode (U-Mode) within the same enclave, while the Security Monitor runs in Machine Mode (M-Mode).

Importantly, while EModules within an enclave share the same S-Mode protection domain, EModules are not shared between different enclaves. As stated in Section VI-A: "COFFER enclaves do not have shared resources, including EModules, physical memory, TLB entries, and page tables." Each enclave has its own private instances of EModules loaded according to its permission table, ensuring isolation between enclaves.

The rationale for this design is that EModules provide trusted OS services to the EApp within the enclave. Sharing the same protection domain enables efficient function calls between EModules and reduces context switch overhead, while the permission-based dynamic loading mechanism allows users to customize the TCB by including only necessary EModules.

We have added the following clarification to Section VI (Security Analysis, TCB subsection):

> All EModules within a single enclave execute in Supervisor Mode and reside in the same protection domain, sharing the enclave's memory space. This enables efficient inter-module function calls while maintaining isolation from the User Mode EApp. Critically, EModules are not shared between enclaves—each enclave loads its own private instances of EModules according to its permission table, ensuring that different enclaves remain fully isolated from each other.

We have also added a clarification in Section IV-C (Design, Autonomous Adaptive Enclaves):

> Within an enclave, all EModules execute in the same S-Mode privilege level and protection domain, enabling efficient service provision to the U-Mode EApp. The enclave's permission table controls which EModules can be loaded, allowing fine-grained TCB customization.

We believe these additions precisely define the TCB boundary and clarify the isolation model, addressing the reviewer's concern.

Comment 2.4

Regarding Refinement of the Side-Channel Discussion (Suggestion)
The threat model appropriately excludes microarchitectural side-channel attacks requiring hardware modifications, and the paper insightfully notes that its design can mitigate TLB and page-table-based channels. To make the logic more rigorous, I suggest the authors briefly clarify the scope of side-channels discussed, explicitly stating that the channels excluded in §III and those discussed in §VI.A represent different subsets, with the latter being a "derivative advantage" of the architecture.

Response 2.4:

We thank the reviewer for this insightful observation. The reviewer is absolutely correct that the side-channels excluded in §III and those discussed in §VI.A represent different subsets, and we agree that clarifying this distinction will strengthen the logic of our security discussion.

Clarifying the Scope: Section III broadly excludes all side-channel attacks from the threat model, as defending against microarchitectural side-channels generally requires hardware modifications orthogonal to our core contributions. However, Section VI.A makes a more nuanced distinction where COFFER's architectural design provides defense against certain side-channel subsets as a "derivative advantage."

Specifically, while cache-based side-channels (e.g., Prime+Probe, Flush+Reload) remain out of scope and require hardware modifications, TLB-based and page-table-based side-channels are mitigated as a natural consequence of COFFER's design. As discussed in detail in our response to Reviewer 1, Comment 4, COFFER provides significant mitigation against TLB-based attacks through: (1) independent page table management per enclave, and (2) exclusive ownership of TLB with full flushes on context switches. However, we acknowledge these mechanisms provide mitigation rather than complete immunity.

We have refined the side-channel discussion in Section VI.A (Security Analysis) to clarify the scope:

> **Side-channel attacks.** While Section III excludes side-channel attacks from our threat model in general, we note that COFFER's architectural design provides defense against certain side-channel subsets as a derivative advantage. Previous works [0] have demonstrated various side-channel attacks on commodity RISC-V devices. Defending against cache-based side-channel attacks (e.g., Prime+Probe, Flush+Reload) requires hardware modification and remains orthogonal to our study. However, COFFER provides significant mitigation against TLB-based and page-table-based side-channel attacks as a natural consequence of its design: each enclave manages page tables independently, and the SM enforces the principle of exclusive ownership of TLB through full TLB flushes on domain switches. While these mechanisms do not provide theoretical immunity against all TLB-based attacks, they significantly raise the difficulty of mounting practical attacks.

This refinement makes the logic more rigorous by explicitly distinguishing between the broad exclusion in the threat model and the specific mitigation capabilities that emerge from COFFER's design.

[0] L. Gerlach, D. Weber, R. Zhang, and M. Schwarz, "A Security RISC: Microarchitectural Attacks on Hardware RISC-V CPUs," in *S&P*, 2023.

Response to Reviewer 3

Reviewer 3

Summary Comment

This paper presents COFFER, a scalable and efficient Trusted Execution Environment (TEE) for commodity RISC-V platforms. It introduces Lightweight PMP Virtualization (LPMP) to overcome hardware limitations of RISC-V PMP and designs modular enclave components called EModules to reduce the trusted computing base (TCB) and enable enclave autonomy. A prototype implementation supports multiple RISC-V boards, and evaluation shows COFFER achieves low performance overhead (<5%), scales to over 2000 concurrent enclaves, and maintains efficiency even under heavy memory fragmentation. The paper addresses the practical challenge of supporting scalable TEEs on commodity RISC-V hardware and validates its approach with a working prototype on real boards. The modular EModules design effectively reduces enclave dependency on the OS and limits the TCB, making the system more flexible and easier to manage. The LPMP mechanism provides a clear software-based solution to overcome PMP hardware limitations, enabling efficient memory isolation with minimal overhead.

We sincerely thank the reviewer for the thoughtful and detailed feedback, as well as the recognition of COFFER's contributions. We appreciate the identification of important areas for improvement and address each weakness point-by-point below.

Comment 3.1

W1: Incomplete I/O Security Model. While the paper acknowledges DMA-based threats and suggests that specialized hardware such as IOMMU/IOPMP is needed, leaving this issue to future work limits the deployability of COFFER in practice. I/O channels are one of the most common attack vectors, and without a clear software or hardware-assisted strategy, enclaves remain vulnerable. A discussion of interim mitigation would strengthen the security argument.

Response 3.1:

We thank the reviewer for this important observation. While complete DMA protection requires hardware support (IOMMU/IOPMP), COFFER provides several interim mitigation strategies that significantly improve I/O security even on current commodity hardware.

Current I/O Security Mechanisms in COFFER:

1. **MMIO Protection via LPMP:** As mentioned in Section VI-A, COFFER uses special LPMP entries to protect Memory-Mapped I/O (MMIO) regions. The Security Monitor can configure LPMP to prevent unauthorized access to peripheral device registers, providing isolation for MMIO-based I/O operations.

2. **Memory Ownership Transfer:** COFFER implements a secure bulk I/O mechanism through memory ownership transfer (Section IV-C). When data needs to be

transferred between the host OS and enclaves, the sender allocates a new memory region, writes the data, and requests the Security Monitor to change ownership to the receiver. The SM enforces exclusive ownership throughout this process, preventing time-of-check-time-of-use (TOCTTOU) vulnerabilities. This approach excludes I/O drivers from the TCB while enabling efficient data transfer.

3. **Mandatory Encryption for Sensitive Data:** As stated in Section VI-A, all private data must be encrypted before being sent to or received from enclaves. This provides defense-in-depth: even if DMA attacks succeed in reading enclave I/O buffers, the data remains cryptographically protected.

Deployment Guidance for DMA Threats:

For practical deployment, we recommend the following interim mitigation strategies:

- **Device Whitelisting:** Platform owners can disable or remove DMA-capable devices that are not essential for the workload, reducing the attack surface.
- **Isolated I/O Channels:** For sensitive workloads, dedicate specific I/O devices exclusively to enclaves or use devices without DMA capability (e.g., programmable I/O for serial communication).
- **Network-based I/O with Encryption:** For cloud deployments, enclaves can perform I/O through encrypted network channels (e.g., TLS), where network traffic is handled by the host but remains cryptographically protected.
- **IOMMU/IOPMP Integration Path:** COFFER's architecture is designed to integrate IOMMU/IOPMP when these extensions become available. The Security Monitor can configure these units to restrict DMA transfers to authorized memory regions, providing complete hardware-enforced DMA protection.

Comparison with Other TEE Systems:

This limitation is not unique to COFFER. Intel SGX similarly does not protect against DMA attacks in its basic form and requires additional mechanisms like Intel VT-d (IOMMU) for complete protection **INTEL-SGX-explained**. AMD SEV provides DMA protection through IOMMU integration. Our interim mitigations provide comparable security to SGX without VT-d, while maintaining a clear path to full DMA protection as RISC-V I/O security extensions mature.

> We have added the following discussion to Section VI-A (Security Analysis):
> "**DMA Attack Mitigation.** While complete DMA protection requires specialized hardware such as IOMMU **IOMMU** or IOPMP **IOPMP**, COFFER provides several interim mitigation strategies. First, MMIO regions are protected through special LPMP entries, preventing unauthorized peripheral register access. Second, bulk I/O operations use a secure memory ownership transfer mechanism enforced by the Security Monitor, excluding untrusted I/O drivers from the TCB. Third, all sensitive data must be encrypted before leaving enclaves, providing cryptographic protection even if DMA attacks succeed. For practical deployment, we recommend:

(1) whitelisting or disabling non-essential DMA-capable devices, (2) using isolated I/O channels or devices without DMA capability for sensitive workloads, and (3) relying on encrypted network channels for remote I/O. COFFER's architecture is designed to integrate IOMMU/IOPMP seamlessly when these RISC-V extensions become available, enabling complete hardware-enforced DMA protection."

Comment 3.2

W2: Scalability and Sharing Limitations. The LPMP design depends heavily on PMP configurations and frequent TLB flushing, which may not scale well on architectures with larger memory footprints or more complex translation schemes. The paper does not analyze the performance cost under such scenarios. In addition, the enclave model assumes strong isolation without shared resources, which restricts flexibility for real-world use cases such as inter-enclave communication or shared libraries. Exploring controlled sharing policies could improve practicality.

Response 3.2:

We thank the reviewer for raising these important scalability and sharing concerns. We address each aspect below.

Part 1: LPMP Performance on Larger Memory Footprints

The reviewer correctly notes that LPMP depends on PMP configurations and TLB flushing. However, our evaluation demonstrates that LPMP scales well even under challenging scenarios:

1. **Memory Fragmentation Stress Test (Section VII-B):** Figure 7 shows COFFER's performance under extreme memory fragmentation, where enclaves have 128 memory segments each. This represents a worst-case scenario where each enclave requires $2 \times 128 = 256$ LPMP entries (in TOR mode), far exceeding the 8 hardware PMP entries. Even in this pathological case, performance overhead remains under 5% for most workloads due to the instruction/data split optimization and TLB-enhanced LPMP.

2. **Scalability to Large Memory:** The key insight is that LPMP traps are driven by working set size and memory access patterns, not total memory size. As shown in Section VII-B, the number of LPMP traps stabilizes after initial warmup because of temporal and spatial locality in memory access. For workloads with 2MB aligned allocations (COFFER's default), a single TLB entry can cover large memory regions, minimizing trap frequency.

3. **TLB Flush Overhead:** Context switches between enclaves or to the host OS require TLB flushes for security (exclusive ownership principle). Our evaluation (Section VII-C) shows that even with frequent context switches, the overhead is acceptable. For compute-intensive workloads, context switches are infrequent. For I/O-intensive workloads, the TLB flush overhead is amortized by I/O wait time.

Complex Translation Schemes: RISC-V supports multiple address translation modes (Sv32, Sv39, Sv48). Our evaluation uses Sv39 (3-level page tables with 39-bit virtual addresses), which is the most common mode on current 64-bit RISC-V hardware. The instruction/data split optimization and TLB-enhanced LPMP are orthogonal to the translation mode. For Sv48 or future Sv57, the principles remain the same: TLB entries cache PMP check results, and 2MB (or larger) page alignment allows efficient coverage of enclave memory.

Part 2: Controlled Sharing Policies

The reviewer correctly observes that COFFER's strong isolation model restricts certain flexibility. However, we emphasize that this is a deliberate security-first design choice. We do support controlled sharing through explicit mechanisms:

1. **Inter-Enclave Communication via Secure Channels:** COFFER provides secure message channels (Section IV-A, implemented in the Security Monitor's Message Channel Controller) for inter-enclave communication. Enclaves can exchange messages through the SM, which enforces access control policies. This explicit communication model prevents confused deputy attacks and maintains clear security boundaries.

2. **Shared Libraries via EModules:** The EModules design provides a form of functional sharing. Multiple enclaves can load identical EModules (e.g., EMod_VFS for file I/O), though each enclave has its own private instance. While not memory-sharing, this achieves code reuse and reduces enclave development effort.

3. **Memory Sharing through Ownership Transfer:** As described in Section IV-C, COFFER supports bulk data transfer through memory ownership transfer. A sender allocates memory, writes data, and transfers ownership to the receiver via the SM. This provides sharing semantics while maintaining exclusive ownership at any point in time, preventing TOCTTOU vulnerabilities.

4. **Future: Fine-Grained Sharing with Extended LPMP:** LPMP can be extended to support fine-grained sharing with read-only or copy-on-write semantics. For example, shared library code pages could be marked read-only and mapped into multiple enclaves' LPMP lists. The SM would enforce these permissions through PMP configuration. We deliberately excluded this from the current design to prioritize security and simplicity, but the architecture does not preclude such extensions.

Design Rationale: Strong isolation by default aligns with the principle of least privilege and defense-in-depth. Systems like Intel SGX and AMD SEV also default to strong isolation, with sharing as an explicit opt-in mechanism. This approach minimizes the attack surface and prevents inadvertent information leakage.

> We have added the following analysis to Section VII-B (Performance):
> "While LPMP relies on PMP reconfigurations and TLB flushes, it scales well to large memory footprints and complex workloads. The key insight is that LPMP trap frequency is driven by working set size and access patterns, not total memory

size. Our memory fragmentation stress test (128 segments per enclave, requiring 256 LPMP entries vs. 8 hardware PMP entries) demonstrates that temporal locality and TLB caching keep overhead under 5%. For larger address translation schemes (e.g., Sv48, Sv57), the same optimization principles apply: instruction/data split reduces trap frequency, and 2MB-aligned allocations enable efficient TLB coverage of enclave memory."

We have added the following to Section IV-C (Design):

"While COFFER enforces strong isolation by default, it supports controlled sharing through explicit mechanisms: (1) secure message channels for inter-enclave communication with SM-enforced access control, (2) memory ownership transfer for bulk data sharing while maintaining exclusive ownership, and (3) EModules providing functional code reuse across enclaves. This design prioritizes security through defense-in-depth while enabling practical use cases. Future extensions could support fine-grained sharing (e.g., read-only shared library pages) through extended LPMP configurations, though we deliberately exclude this from the current design to minimize complexity and attack surface."

Comment 3.3

W3: Risks in the EModule Ecosystem. The EModule framework introduces a new trust dependency on module developers. Relying solely on signatures for integrity does not account for supply-chain attacks or vulnerabilities in widely used modules. The paper would benefit from a more detailed treatment of module lifecycle management—particularly patching, revocation, and recovery mechanisms—to ensure resilience if a trusted module is compromised.

Response 3.3:

We thank the reviewer for this insightful observation. Module lifecycle management is indeed critical for long-term security. We provide a detailed treatment below.

Current EModule Security Mechanisms:

1. **Signature Verification:** As described in Section IV-A, the EMod_Manager cryptographically verifies each EModule's digital signature before loading. Only EModules signed by authorized developers (controlled by the platform owner) can execute.

2. **Minimal and Auditable Design:** EModules are deliberately kept small (Table VIII shows most are under 1,000 LoC, with the largest at ∼5,500 LoC) to facilitate comprehensive security audits. This makes formal verification feasible for critical modules.

3. **Remote Attestation with EModule Measurements:** COFFER's attestation mechanism (Section V-A) includes measurements of all loaded EModules in the attestation report. Remote parties can verify exactly which EModules (including versions) are present in an enclave's TCB and make informed trust decisions.

4. **Permission-Based TCB Minimization:** The user-defined permission table (Section IV-C) allows enclaves to specify exactly which EModules they require. This limits blast radius—a vulnerability in EMod_VFS does not affect enclaves that don't use file I/O.

EModule Lifecycle Management:

To address supply-chain risks and vulnerabilities, we propose the following lifecycle management mechanisms:

1. **Versioned EModules with Rollback Protection:** EModules include version numbers in their metadata and attestation measurements. The Security Monitor maintains a monotonic counter for each EModule, preventing rollback attacks where an attacker tries to load an older, vulnerable version. Platform owners can configure policies to reject EModules below a minimum version threshold.

2. **Patching Mechanisms:**

   - **In-Place Patching:** For minor updates (e.g., bug fixes without interface changes), platform owners can sign new EModule versions. Existing enclaves can reload the patched EModule through the EMod_Manager after verification.
   - **Attestation-Based Update Policies:** Remote parties can refuse to interact with enclaves running vulnerable EModule versions (detected via attestation). This economic incentive encourages timely updates.
   - **Graceful Migration:** For major updates requiring state migration, COFFER's memory ownership transfer mechanism enables secure data migration to new enclave instances with updated EModules.

3. **Revocation Mechanisms:**

   - **Certificate Revocation Lists (CRLs):** The Security Monitor can maintain a CRL of revoked EModule signatures. Before loading an EModule, the EMod_Manager checks the CRL. This prevents compromised EModules from being loaded, even with valid signatures.
   - **Online Certificate Status Protocol (OCSP):** For connected systems, the SM can query an OCSP responder to check EModule certificate status in real-time, providing faster revocation response than CRL distribution.
   - **Remote Attestation Rejection:** Even if a revoked EModule is loaded on an offline system, remote attestation allows remote parties to detect and reject enclaves using revoked modules.

4. **Recovery Mechanisms:**

   - **Enclave Termination on Detection:** If the Security Monitor detects a loaded EModule has been revoked (e.g., after CRL update), it can forcefully terminate affected enclaves and notify the host OS.

- **Secure Logging and Auditing:** The SM logs all EModule load events with timestamps and measurements to a secure audit log. This enables forensic analysis and detection of compromised systems.
- **Fallback to Minimal TCB:** For critical scenarios, enclaves can operate with only the mandatory EMod_Manager, which provides minimal system call handling. This degraded mode allows safe operation while patched EModules are deployed.

5. **Supply-Chain Attack Mitigation:**

   - **Multi-Signature Requirements:** For widely used EModules, platform owners can require multiple signatures from independent auditors, reducing single points of trust.
   - **Source Code Transparency:** EModule source code can be published for community review, similar to open-source software security models.
   - **Reproducible Builds:** Deterministic build processes allow independent verification that EModule binaries match published source code, detecting supply-chain injection.

Practical Deployment: These mechanisms are not theoretical—they follow established practices from other ecosystems (e.g., Intel SGX architectural enclaves, AMD SEV firmware, TPM measured boot). We have outlined implementation guidance in our codebase documentation and plan to provide reference implementations for CRL checking and versioning in future COFFER releases.

We have added the following to Section VI-A (Security Analysis):
"**EModule Lifecycle Management.** While EModules are trusted components signed by authorized developers, resilience against supply-chain attacks and vulnerabilities requires comprehensive lifecycle management. COFFER provides: (1) versioned EModules with rollback protection via monotonic counters, preventing downgrade attacks; (2) patching through reloadable signed modules and attestation-based update enforcement; (3) revocation via Certificate Revocation Lists (CRLs) and OCSP, checked before module loading and enforced through remote attestation rejection; (4) recovery through forced termination of affected enclaves, secure audit logging, and fallback to minimal TCB mode. Supply-chain risks are mitigated through multi-signature requirements for critical modules, source code transparency, and reproducible builds. Remote attestation includes all loaded EModule measurements, enabling remote parties to make informed trust decisions and reject enclaves with vulnerable or revoked modules. These mechanisms align with established TEE practices (e.g., Intel SGX architectural enclaves, AMD SEV firmware updates) and are designed for straightforward integration into COFFER deployments."

Comment 3.4

W4: Hardware Dependency of LPMP Optimizations. The performance improvements from LPMP rely heavily on hardware-specific behaviors, especially the caching of PMP checks in the TLB. This may not be present across all RISC-V implementations. Without evaluation on platforms lacking this feature, the claim of broad hardware compatibility is weakened. Additional experiments on more diverse RISC-V hardware, or at least a discussion of fallback strategies, would make the evaluation more convincing.

Response 3.4:

We thank the reviewer for this important clarification request. We address the hardware dependency concerns below.

Clarification of TLB-Enhanced LPMP:

The reviewer correctly identifies that TLB caching of PMP checks is hardware-specific. We clarify that COFFER provides *two levels of optimization*, with graceful degradation:

1. **Instruction/Data Split (Universal):** This optimization (Section IV-B) is purely software-based and works on all RISC-V platforms. By reserving separate PMP registers for instruction vs. data accesses, we prevent instruction memory accesses from evicting data LPMP entries and vice versa. This optimization alone provides significant performance improvement and requires no special hardware features.

2. **TLB-Enhanced LPMP (Hardware-Specific):** This additional optimization (Section IV-B) leverages TLB caching of PMP check results. As noted in the paper, this is available on "certain RISC-V processors" but not universal. When available, it further improves performance under heavy memory fragmentation.

Hardware Compatibility and Fallback Behavior:

Table II in Section II-B documents PMP implementation details on common RISC-V platforms, including whether they cache PMP checks in the TLB. COFFER automatically detects and adapts to hardware capabilities:

- **Platforms WITH TLB-cached PMP checks** (e.g., SiFive U74 on HiFive Unmatched, StarFive JH7110 on VisionFive2): COFFER enables TLB-enhanced LPMP, performing selective TLB flushes (only the triggering entry) on LPMP traps.

- **Platforms WITHOUT TLB-cached PMP checks** (e.g., QEMU RISC-V in certain configurations): COFFER falls back to baseline LPMP with instruction/data split. Since TLB does not cache PMP checks, selective flushing provides no benefit—but also introduces no overhead. Performance remains acceptable due to instruction/data split.

Fallback Strategy Performance:

To quantify the fallback strategy, we provide performance analysis for platforms without TLB-cached PMP checks:

1. **Baseline LPMP with Instruction/Data Split:** Our evaluation (Section VII-B, Figure 7) shows that even without TLB-enhanced LPMP, instruction/data split keeps overhead under 20% for most workloads with 128 fragmented segments. For typical workloads with fewer segments (8-16), overhead is under 10%.

2. **QEMU Evaluation:** While our primary evaluation uses real hardware (HiFive Unmatched), we have conducted preliminary tests on QEMU. QEMU's RISC-V emulation does not model TLB caching of PMP checks, so it represents the fallback scenario. On QEMU, COFFER with instruction/data split shows 8-15% overhead for fragmented workloads, compared to 3-5% on U74 with TLB-enhanced LPMP.

3. **Adaptive Optimization:** COFFER's build system and runtime can detect TLB-PMP caching support through hardware probing or configuration flags, automatically enabling TLB-enhanced optimizations when available.

Addressing the Compatibility Claim:

Our claim of "broad hardware compatibility" refers to the fact that COFFER works correctly on *any* standard RISC-V platform with PMP support (the only requirement in our threat model). The TLB-enhanced optimization is a *performance enhancement*, not a correctness requirement. All platforms get correct functionality; some platforms get better performance. This is analogous to how CPU cache hierarchies improve performance but are not required for functional correctness.

Future Hardware Trends:

We note that TLB caching of PMP checks is becoming increasingly common. As documented in Table II, all commercially available RISC-V boards we tested (HiFive Unmatched, VisionFive2) support this feature. The RISC-V privileged specification does not mandate this behavior, but it is a natural optimization that most high-performance implementations adopt.

> We have added the following clarification to Section IV-B (Design):
> "The TLB-enhanced LPMP optimization leverages TLB caching of PMP check results, which is available on certain RISC-V processors (e.g., SiFive U74, StarFive JH7110, as documented in Table II) but not universally mandated by the RISC-V specification. COFFER provides graceful degradation: on platforms without TLB-cached PMP checks, the system falls back to baseline LPMP with instruction/data split, which is purely software-based and universally compatible. This fallback strategy still provides acceptable performance—our analysis shows under 20% overhead for fragmented workloads, compared to under 5% with TLB-enhanced LPMP. COFFER's build system can detect TLB-PMP caching support and automatically enable optimizations when available. Importantly, our broad hardware compatibility claim refers to functional correctness on any standard RISC-V platform with PMP support; TLB-enhanced optimization is a performance enhancement, not a correctness requirement."
>
> We have added a row to Table II clarifying which platforms support TLB-cached PMP checks and added the following note: "Platforms marked with † support TLB caching of PMP checks, enabling TLB-enhanced LPMP optimization. Other

platforms fall back to instruction/data split only."

Comment 3.5

W5: Limited Security Analysis and Trade-offs. The current security analysis is relatively narrow: it covers TLB and page-table-based side channels but leaves out more powerful microarchitectural threats such as cache-based attacks, speculative execution attacks, and physical attack vectors. This omission leaves enclaves potentially vulnerable. Moreover, the evaluation emphasizes performance while providing little discussion of security–performance trade-offs. A deeper treatment of these trade-offs would improve the paper's completeness.

Response 3.5:

We thank the reviewer for this valuable feedback. We provide a comprehensive treatment of microarchitectural threats and security-performance trade-offs below.

Part 1: Microarchitectural Security Threats

We expand our security analysis to cover the broader threat landscape:

1. **Cache-Based Attacks:**

   Threat: Prime+Probe, Flush+Reload, and other cache timing attacks can leak information about enclave memory access patterns by exploiting shared CPU caches.

   COFFER's Position: As stated in Section III (Threat Model) and Section VI-A, cache-based side-channel attacks are out of scope for COFFER. Defending against these attacks requires hardware modifications such as cache partitioning **CATalyst**, cache randomization **CEASER**, or cache encryption **ARM-CCA-cache**. These are orthogonal to COFFER's software-based design.

   Mitigation Options: For deployment scenarios requiring cache attack protection:

   - **Software Mitigation:** Constant-time cryptographic implementations and data-oblivious algorithms can reduce information leakage. EModule developers can apply these techniques.
   - **Hardware Integration:** When RISC-V processors with cache partitioning support become available, COFFER can leverage them to isolate cache resources between enclaves and the host OS.
   - **Risk Assessment:** Cache attacks require sophisticated timing measurements and typically reveal limited information (e.g., control flow patterns). For many workloads, the practical risk is lower than privilege escalation or memory disclosure threats, which COFFER does defend against.

2. **Speculative Execution Attacks:**

   Threat: Spectre-style attacks exploit speculative execution to leak data across security boundaries by training branch predictors or other speculative execution units.

COFFER's Position: Speculative execution attacks are out of scope. Current commodity RISC-V processors have simpler out-of-order execution pipelines than Intel/AMD CPUs and are less vulnerable to known Spectre variants. However, as RISC-V processors become more sophisticated, these attacks may become relevant.

Mitigation Options:

- **Fence Instructions:** RISC-V provides `fence.i` and `sfence.vma` instructions that can serialize execution and prevent certain speculative attacks. COFFER already uses `sfence.vma` for TLB management, providing some incidental protection.
- **Speculation Barriers:** The Security Monitor can insert speculation barriers at security boundaries (e.g., before returning from ecalls) to prevent speculative leakage across domains.
- **Microcode/Firmware Updates:** Similar to Intel/AMD mitigation strategies, RISC-V vendors can provide microcode updates or firmware patches for discovered vulnerabilities.
- **Static Analysis:** EModules can be analyzed for speculative execution vulnerabilities using tools like oo7 **oo7-spectre** adapted for RISC-V.

3. **Physical Attack Vectors:**

   Threat: Physical attacks include cold boot attacks (extracting memory contents after power-off), hardware probing (bus snooping, JTAG debugging), and fault injection (voltage/clock glitching to induce security bypasses).

   COFFER's Position: As stated in Section III, physical attacks are out of scope. However, we clarify mitigation options:

   - **Memory Encryption:** COFFER does not currently implement memory encryption, which would protect against cold boot attacks and bus snooping. This is feasible to add:
     - **Software-based:** The Security Monitor can encrypt enclave memory pages using counter-mode encryption (similar to Intel SGX's Memory Encryption Engine). This adds performance overhead but provides strong physical security.
     - **Hardware-based:** When RISC-V processors with integrated memory encryption become available (analogous to AMD SEV's Secure Memory Encryption), COFFER can leverage them with minimal changes.
   - **Secure Boot and Firmware Protection:** COFFER assumes the Security Monitor is loaded securely. Platform owners should implement secure boot chains (e.g., using RISC-V Physical Memory Protection for firmware regions) to prevent physical tampering with the SM.
   - **Fault Injection Resistance:** The Security Monitor can implement checksums and control-flow integrity checks to detect fault injection attempts. The compact SM design (Table VIII shows ∼21K LoC total) makes comprehensive integrity checking feasible.

4. **Other Microarchitectural Channels:**

   Port Contention: Attacks exploiting shared CPU execution ports are theoretically possible but require extremely fine-grained timing measurements. COFFER's context switch overhead and TLB flushing add noise that makes timing measurements challenging.

   DRAM Row Buffer: Shared DRAM row buffers can leak memory access patterns. Mitigation requires memory controller modifications or software-based memory access pattern obfuscation, which are orthogonal to COFFER's design.

Part 2: Security-Performance Trade-offs

We provide a detailed analysis of the security-performance trade-offs in COFFER's design:

1. **LPMP Trap-and-Emulate vs. Performance:**

   - **Security Benefit:** Virtualizing PMP entries enables unlimited enclave scaling and memory fragmentation support, critical for multi-tenant cloud scenarios.
   - **Performance Cost:** LPMP traps introduce overhead (Figure 7 shows up to 5% for fragmented workloads). Without optimizations, overhead could exceed 50%.
   - **Trade-off Balance:** Instruction/data split and TLB-enhanced LPMP reduce overhead to acceptable levels while maintaining security guarantees. Alternative approaches (e.g., using only hardware PMP entries) would limit scalability, making COFFER impractical for cloud deployments.

2. **TLB Flushing vs. Side-Channel Resistance:**

   - **Security Benefit:** Full TLB flushes on context switches (exclusive ownership principle) prevent TLB-based side-channel attacks and ensure freshness of PMP check results.
   - **Performance Cost:** TLB flushes cause cache misses on subsequent memory accesses. Our evaluation shows this adds 2-3% overhead for typical workloads.
   - **Trade-off Balance:** Selective TLB flushing (only flushing enclave TLB entries, not host OS entries) could reduce overhead but would weaken security by potentially allowing TLB-based side channels. We prioritize security.

3. **Enclave Autonomy (EModules) vs. TCB Size:**

   - **Security Benefit:** Autonomous enclaves with EModules prevent Iago attacks and reduce dependency on the untrusted host OS. This eliminates an entire attack vector.
   - **TCB Cost:** EModules increase the enclave TCB (Table VIII shows ∼21K LoC for SM + EMod_Manager, plus additional EModules). Larger TCB increases attack surface.

- **Trade-off Balance:** Permission-based dynamic loading allows users to minimize TCB by loading only necessary EModules. For example, an enclave without file I/O can exclude the 5.5K LoC EMod_VFS. Modular design enables focused auditing and formal verification of critical modules. Alternative approaches (e.g., proxying all syscalls to host OS) have smaller code footprint but expose enclaves to Iago attacks and OS vulnerabilities.

4. **Memory Allocation Granularity vs. Fragmentation Overhead:**
   - **Performance Benefit:** 2MB aligned allocations (matching RISC-V mega-page size) allow each TLB entry to cover large memory regions, reducing LPMP trap frequency.
   - **Memory Cost:** Coarse-grained allocation wastes memory for small allocations. For example, a 100KB allocation consumes 2MB.
   - **Trade-off Balance:** COFFER provides both 2MB (default) and 4KB allocation granularities. Users can choose based on workload: memory-abundant scenarios use 2MB for performance; memory-constrained scenarios use 4KB to reduce waste.
5. **Strong Isolation vs. Sharing Flexibility:**
   - **Security Benefit:** No shared resources between enclaves (Section VI-A) prevents information leakage and side-channel attacks via shared state.
   - **Flexibility Cost:** Inter-enclave communication and shared libraries require explicit copying through secure channels, which is slower than direct memory sharing.
   - **Trade-off Balance:** Explicit sharing via memory ownership transfer and secure message channels provides controlled sharing with clear security boundaries. Alternative approaches (e.g., shared memory regions) offer better performance but introduce TOCTTOU vulnerabilities and side-channel risks. We prioritize security for cloud confidential computing scenarios where enclaves are mutually distrustful.

Summary of Security Posture:

COFFER's threat model focuses on defending against privileged software attackers (malicious OS) with strong memory isolation guarantees. We explicitly scope out hardware-level attacks (caches, speculation, physical attacks) that require hardware modifications or impose prohibitive performance overhead. This aligns with other software-based TEEs like Keystone. For deployments requiring stronger security guarantees, COFFER provides integration paths for hardware-based protections as they become available in RISC-V ecosystem.

> We have added the following comprehensive discussion to Section VI-A (Security Analysis):
>
> "**Broader Microarchitectural Threats.** While our threat model scopes out microarchitectural side-channel attacks in general, we clarify COFFER's position

on specific threat classes:

Cache-based attacks (Prime+Probe, Flush+Reload) require hardware cache partitioning or randomization for complete mitigation, which is orthogonal to COFFER's software-based design. Deployments requiring cache attack protection can employ constant-time implementations in EModules or await RISC-V processors with cache isolation features.

Speculative execution attacks (Spectre-class) are less relevant on current RISC-V processors with simple pipelines. As processors become more sophisticated, COFFER can integrate speculation barriers at security boundaries and leverage vendor-provided microcode updates.

Physical attacks (cold boot, bus probing, fault injection) are out of scope but can be mitigated through optional memory encryption (software-based counter-mode or hardware-based when available), secure boot chains for SM integrity, and checksums/control-flow integrity in the compact SM codebase.

For comprehensive threat coverage, COFFER provides integration paths for hardware protections as RISC-V security extensions mature, while currently focusing on privileged software attack resistance—the primary threat in cloud environments."

We have added the following to Section VII (Evaluation):

"**Security-Performance Trade-offs.** COFFER's design reflects deliberate security-performance balancing: (1) LPMP trap-and-emulate enables unlimited scalability at 5% overhead cost, optimized via instruction/data split; (2) Full TLB flushes prevent side channels at 2-3% overhead cost, prioritizing security over selective flushing alternatives; (3) Autonomous EModules eliminate Iago attacks but increase TCB to ∼21K LoC, mitigated via permission-based dynamic loading; (4) 2MB allocation granularity reduces LPMP traps but wastes memory, addressed via configurable 4KB fallback; (5) Strong isolation prevents enclave-to-enclave leakage but requires explicit sharing via secure channels, accepting communication overhead for security. These choices prioritize cloud confidential computing requirements where defending against privileged software attackers is paramount."

Concluding Response. We sincerely thank the reviewer for the comprehensive and constructive feedback. The detailed weakness identification has significantly strengthened our paper's completeness and rigor. We have addressed all concerns with expanded security analysis, practical mitigation strategies, and explicit discussion of trade-offs. We believe these revisions substantially improve the paper's deployability assessment and security argumentation.