Response Letter of Paper #283

Dear ASE'25 Program Committee Members and Reviewers:

Regarding our previous submission entitled "VRExplorer: A Model-based Approach for Semi-Automated Testing of Virtual Reality Scenes" to your ASE 2025, we received the following scores: 3 (Weak accept), 2 (Weak reject), and 3 (Weak accept) from three reviewers. The metareview recommendation is **Major Revision**. Many thanks for your constructive comments and positive evaluations of our paper. Let us express our sincere thanks to all the reviewers and PC members. Your constructive comments guide us to further improve this paper in terms of novelty and soundness.

We have carefully revised the paper by addressing all of your concerns. In this response letter, we first list your comments in a point-by-point manner. We then present our responses accordingly. Moreover, we also present a highlighted version of the revised paper for your reference so that you can quickly identify our revised parts.

Your sincerely,

Authors

Point-by-Point Response to Metareview

This paper first adapted the model-based approach to test VR applications and presented promising experimental results compared with existing works. After discussing the authors' responses, the reviewers still have some concerns about the effort required for the manual model abstraction process and whether it can be generalized. To address these concerns, a major revision including the following changes is requested.

* The revised paper should include 2-3 additional apps from a different domain as subjects of evaluation to show that the work is generalizable.

Response: Thanks for your comment.

To further demonstrate the generalizability of our approach, we have added two additional VR applications from two different domains, such as Strategy Board Game and Medical Care.

In particular, *VRChess* is a *Strategy Board Game* type of VR applications, in which players manipulate chess pieces via XR-specific interactions. It encompasses various interaction types, including Grab/Release/Throw/Place, Classical Software GUI interactions, and XR GUI interactions. On this VR app, our approach still achieves 71.74% ELOC Coverage and 87.72% Method Coverage compared to the baseline's 10.74% and 50.88%, respectively.

Parkinson-VR is a Medical-type VR application designed to provide Parkinson's patients with immersive environments. It includes diverse VR tasks, such as Ray Interactor-based interactions. On this VR app, our approach achieves 95.65% ELOC Coverage and 100% Method Coverage compared to the baseline's 42.03% and 53.85%, respectively.

These additional evaluations, reported in the revised version of the paper, further demonstrate the generality and effectiveness of our method across different VR domains.

* The revised paper should include a quantitative measurement of the manual model abstraction effort (e.g., the time of manual app exploration, number of interactions manually extracted, and sizes of abstracted models).

Response: Thanks for your comment.

In the revised paper, we supplemented more details about the quantitative measurement of the manual model abstraction. We excerpt the revised parts in the following perspectives:

* **The time of manual app exploration:**

In Section IV-A (Implementation):
"**Implementation of Model Abstraction.** … Manual exploration per project ranged from 30 to 90 minutes, depending on the engineer's proficiency."

*** Number of interactions manually extracted:**
In Section IV-A (Implementation):

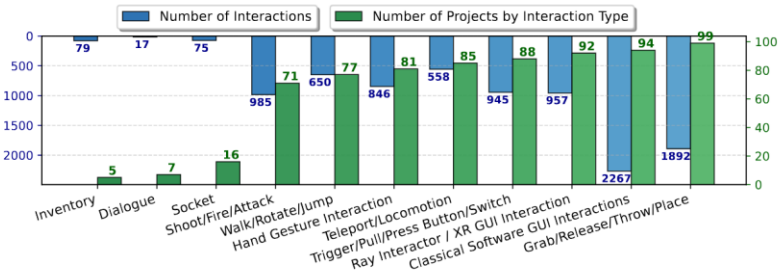

Fig. 6: Distribution of Interactions in Dataset.

"Fig. 6 indicates that the dataset includes both prevalent interactions (e.g., grabbing, GUI operations, and teleportation) and infrequent ones (e.g., inventory and dialogue). (e.g., inventory and dialogue). Together, these interactions cover all major interaction tasks identified in [28], thereby highlighting the constructed dataset's representativeness and diversity."

[28] J. Hertel, S. Karaosmanoglu, S. Schmidt, J. Br¨aker,M. Semmann, and F. Steinicke, "A taxonomy of interaction techniques for immersive augmented reality based on an iterative literature review," in 2021 IEEE International Symposium on Mixed and Augmented Reality (ISMAR),2021, pp. 431–440.

*** Statistical Metrics of Dataset Projects:** We have supplemented details about quantitative metrics in terms of scenes/objects/assets of dataset projects.

We excerpted the revised parts as follows:

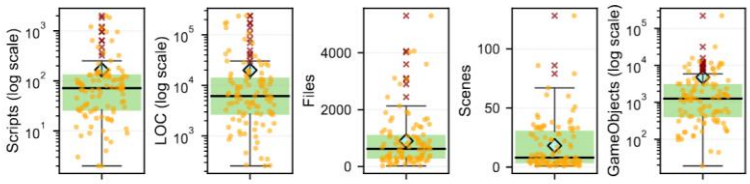

Fig. 5: Boxplots of scripts, LOC, files, and scenes.

TABLE IV: Statistical Metrics of Collected 102 Projects.

Metric	Mean	Variance	Min	Q1	Median	Q3	Max
Scripts	166.45	110,953.52	2	26.25	71.50	130.25	2,004
LOC	19,910.85	1,851,165,810.13	256	2,707.25	6,129.00	13,762.75	237,725
Files	902.07	902,718.22	21	301.25	620.50	1,090.00	5,303
Scenes	18.12	455.06	1	4.00	8.00	30.25	128

In Section IV-A (Implementation):
"**Project Collection and Analysis.** We implemented a crawler to collect Unity-based VR GitHub projects with keywords like "VR", "AR", "unity", and "xr". From 971 initially filtered projects, we manually performed quality checks (e.g., compilation errors and version conflicts), consequently retaining 102 high-quality projects.

We then analyze the structural and interaction characteristics of all projects in the dataset to assess their representativeness and complexity. Table IV shows varied project sizes. While most projects are relatively small (medians containing 71.5 scripts, 6,129 LOC, 620 files, and 8 scenes), a few of them with large sizes markedly raise medians. Fig. 5 plots their distribution. Fig. 6 indicates that the dataset includes both prevalent interactions (e.g., grabbing, GUI operations, and teleportation) and infrequent ones (e.g., inventory and dialogue). (e.g., inventory and dialogue). Together, these interactions cover all major interaction tasks identified in [28], thereby highlighting the constructed dataset's representativeness and diversity."

*** Sizes of abstracted models:** Regarding the sizes of abstracted models, in our method, the size of the abstracted model is represented/implemented mainly by the number of scripts obtained in the EAT framework. Therefore, we have added more details in Section IV-A, part of "Implementation of the Model Abstraction".

We excerpted the revised parts as follows:

In Section IV-A (Implementation):
"**Implementation of Model Abstraction.** Based on the above analysis, we extract abstract models using derived heuristics. The implementation comprises six scripts in the Action layer, four scripts in the Entity layer, and four pre-defined Mono scripts. Additionally, five pre-defined task models orchestrate these components, resulting in more than 1,600 lines of code in total. Manual exploration per project ranged from 30 to 90 minutes, depending on the engineer's proficiency."

* In the title, "Automated" should be changed to "Semi-Automated" to better reflect the content of the paper.

Response: Thanks for your comment. We have revised the title accordingly, changing *"Automated"* to *"Semi-Automated"* to better reflect the content of the paper.

Point-by-Point Response to Reviewer A

Paper summary

The rapid expansion of Virtual Reality (VR) applications brings in unique challenges in testing due to complex 3D environments and diverse interactions. This paper proposes VRExplorer, a novel model-based testing tool, to address these challenges with its Entity, Action, and Task (EAT) framework, enabling effective interaction with virtual objects and systematic scene exploration. VRExplorer also incorporates a path-finding algorithm and a Probabilistic Finite State Machine (PFSM) to achieve higher coverage and efficiency. The evaluation validated the enhancement over existing approaches on code coverage and bug detection.

Strengths
+ Testing of VR applications is an important software engineering problem in the emerging domain
+ Comparison with the SOTA approach on multiple datasets and achieve better results
+ Supporting multiple VR-specific testing operations such as grabbing

Weaknesses
- The ablation study considers only removing interactions
- Identified bugs seem to have not been validated by the developers

Response: Many thanks for your positive comments on this paper.

Regarding your concerns, we present the detailed response to your comments as follows.

Detailed comments for authors

Significance: Testing of AR scenes is a very important problem in software engineering of virtual reality, an emerging area in computer science. Currently, the approaches in the area are still in preliminary stage and missing a lot of support to VR-specific interactions. The paper filled this important gap and the evaluation shows that the proposed work can achieve high coverage. The implementation is based on Unity, which is a dominating framework in the area, so it is expected that the technique and the tool can be potentially adopted by VR developers.

Novelty: The problem of VR testing and the missing support of VR-specific interactions such as grabbing are well known in the area. However, this is the first work on automatic VR testing to fill this gap. The model-based testing strategy and the path finding technique are also used before in automatic testing, but they are also the first time being adapted to the VR testing area. The dataset which incorporates an existing dataset and includes more projects is also a good contribution to the area.

Soundness: The comparison of the proposed approach with SOTA is thorough and on multiple datasets. So I believe the improvement is real and significant. The ablation study could use some more configurations to help understand the effectiveness of its component better. In particular, the current ablation study removes only different interactions in the configuration of variants,

but does not consider the variants other components such as navigation algorithms and the information extraction component. It is good to see that the approach is able to find additional real-world bugs. However, I do not find in the paper how these bugs were handled by the developers. Have they been confirmed or fixed by the developers?

Response: Thanks for your comments. We present the detailed response to each of your comments in the following part.

Verifiability: The paper has made available the code and project dataset being used in the evaluation. So I believe the work is reproducible.

Presentation: The paper is easy to read and the approach / evaluation sections are well structured.

Questions for authors' response
1. Why only interactions are removed in the ablation study but not other components?

Response: Thanks for your positive comment.

Regarding your concern about the ablation study, we present the detailed explanation as follows.

Our ablation study mainly focuses on the interaction modules because VRExplorer is built upon the EAT framework, which decomposes VR interactions into reusable action units. As the core of the exploration process, *interactions* represent the components that can be meaningfully isolated. In contrast, other modules are tightly coupled with the pipeline and act as prerequisites for system functionality, making their independent removal infeasible.
We excerpted the revised parts as follows:

In Section V-B (Ablation Study):
"To assess the contribution of different components, we perform an ablation study by selectively removing modules from the proposed *VRExplorer*. Given that *VRExplorer* is developed on top of the EAT framework, which decomposes VR interactions into reusable action units. Thus, EAT serves as the core exploration task of inherently focusing on interacting with objects. As a result, our ablation primarily targets interaction-related components within EAT. The EAT framework constitutes the key methodological contribution of *VRExplorer* and is the only component that can be meaningfully isolated while preserving system functionality. In contrast, other modules and components serve as indispensable prerequisites for enabling exploration in VR scenes and are tightly integrated into the overall pipeline, rendering their removal infeasible for independent evaluation."

To further address your concern, we also conducted additional ablation experiments to evaluate VRExplorer's *navigation* and *pathfinding* algorithms.

Compared the Greedy algorithm (used as baseline) with a Backtracking-and-Pruning algorithm, which formulates the navigation task as a TSP problem and computes a Hamiltonian path for globally optimal traversal. Results show that the TSP-based approach improves runtime efficiency by 17.5% (1494.41s vs. 1233.70s across 100 randomized rounds with up to 100

interactable objects), though it also incurs higher computational cost and reduces frame rates. The reduced frame rates may degrade VR performance.

In the revised paper, we supplemented additional details to clarify these issues. We excerpted the revised parts as follows:

In Section VI-C (Discussion on Path-finding Algorithms):
"We also revisit the design choice of adopting the Greedy algorithm for navigation. As described in Section III, *VRExplorer* supports both a Greedy strategy and a Backtracking-with-Pruning (B&P) algorithm. The latter solves the navigation task as a Traveling Salesman Problem (TSP) to compute globally optimal Hamiltonian paths. To quantify their differences, we performed additional experiments on randomized tasks with up to 100 interactable objects. Results show that the B&P algorithm achieves a 17.5% improvement in runtime efficiency (1233.70s vs. 1494.41s over 100 rounds) over the Greedy baseline. However, this improvement comes at the cost of significantly higher computational overhead, leading to reduced frame rates. This trade-off confirms our design decision: while the B&P algorithm can provide globally optimal solutions, the Greedy algorithm remains more practical for maintaining smooth interactive performance in real-time testing."

2. What happened to the detected real-world bugs?

Response: Thanks for your question.

We validated all three reported bugs using multiple, independent evidence sources (Unity console logs, runtime exception traces, and script-trigger chains) to ensure they are real-world bugs rather than false positives. Concretely:

- The bug in *UnityCityView* was confirmed and subsequently fixed by the original developers (this is evidenced by subsequent commits in the project repository).
- For the two bugs in *EscapeGameVR*, we performed root-cause analyses to demonstrate their validity:
  - **Functional bug:** an *Unassigned Reference Exception* triggered when a serialized field (`arrowSpawnPoint` in `ArrowController`) is accessed without being assigned to the Inspector. This is consistent with CWE-395 (null-dereference) and matches CodeQL rules for improper initialization.
  - **Non-functional bug:** caused by a missing `ArrowPrefab` resource, which prevents instantiation of the bow; such missing-resource issues are well known in Unity prefab usage.
- We reported the *EscapeGameVR* issues to the original developers via email; to date, we have not received a response.

In the revised paper, we supplemented additional details to clarify these issues. We excerpted the revised parts as follows:

In Section V-C (Bug Detection in VR Projects):
"**Detected Bugs Validity Confirmation.** To further confirm the validity of detected bugs, we validated all reported issues through Unity console logs, runtime exception traces, and script-

trigger chains. Among them, the bug in *UnityCityView* has been confirmed and fixed by its developers (evidenced by subsequent commits). For the two bugs in *EscapeGameVR*, we performed root-cause analyses: the functional bug is an *Unassigned Reference Exception*, consistent with CWE-395 [35] and CodeQL's rule on improper initialization [36], while the non-functional bug is caused by a missing ArrowPrefab resource, which is a common Unity issue documented in the official Prefab guidelines [37]. We also reported these issues to the original developers of EscapeGameVR, but no response was received till now."

[37] Unity Technologies, "BrokenPrefabAsset," Unity Documentation, https://docs.unity3d.com/6000.1/Documentation/ScriptReference/BrokenPrefabAsset.html, 2025.

Artifact check

3. Satisfactory

Artifact check comments (if any)

The code and project repositories used in the evaluation are both available. I did not try to set up the tool but the code looks reasonable in a quick scan.

Response: Many thanks for your time reading this paper. Thanks for your positive evaluation of the artifact.

Point-by-Point Response to Reviewer B

Paper summary

The paper presents a model-based testing framework, VRExplorer, for testing virtual reality (VR) applications. The model underlying VRExplorer is constructed based on three key abstractions: VR Entity, Action, and Task. Building on these models, the approach employs a path-finding algorithm and a probabilistic state machine to thoroughly test Unity-based VR applications through in-depth scene exploration and comprehensive interaction with virtual objects. Experimental results show that VRExplorer outperforms the state-of-the-art approach, VRGuide, achieving average performance gains of 72.8% in executable lines of code (ELOC) coverage and 46.9% in method (function) coverage.

Strengths
+ Significance (significant for VR test scene coverage)
+ Verifiability (Available replication package)
+ Presentation (easy to read)

Weaknesses
- Soundness (potential bias in evaluation dataset selection)
- Novelty (modeling process seems more manual)

Response: Many thanks for your positive evaluation and constructive comments on this paper. To address your concerns, we have provided the following point-by-point responses.

Detailed comments for authors

Significance: Developing testing tools for VR applications is both essential and timely. Given that testing VR applications requires sufficient scene coverage and interaction fidelity, tools like VRExplorer are undoubtedly beneficial for helping developers test their applications more effectively and efficiently.

Soundness:
- In Section III.A, the paper discusses Project Collection and Analysis. However, the analysis process appears to be only partially automated and largely static in nature. The approach relies heavily on static inspection of scene hierarchies and attached scripts, which raises concerns about its ability to handle dynamic behaviors. Specifically, dynamically instantiated objects or components added at runtime through methods like Instantiate() or AddComponent() may be missed. Moreover, the paper provides only a brief description of the analysis procedure, lacking clear details on how C# scripts and their associated GameObjects are analyzed. A more thorough explanation of the analysis methodology would strengthen the technical soundness of the approach.

Response: Thanks for your comment.

Let us further explain the key innovations of this paper and clarify your concerns about soundness as follows.

We agree with you that relying only on static inspection may miss some runtime behaviors such as objects created by `Instantiate()` or components added via `AddComponent()`. Our approach, however, is a two-pass pipeline that combines (i) *static inspection* of scenes, prefabs, and C# scripts and (ii) *dynamic frame-by-frame runtime observation* with lightweight instrumentation. In particular, at runtime, we snapshot and differentiate the active GameObjects from their components, then record UnityEvent listeners (both persistent and runtime-attached), and trace interaction entry points with event dispatches. This process allows us to capture dynamically instantiated objects and runtime component attachments that are invisible to scene files. Both the static inspection and the dynamic process are consolidated to produce the Object and Action abstractions, which are then fed into our EAT framework (interface candidates), PFSM behavior space (task vs. exploration), and the configurable UnityEvent list used by *VRExplorer*.

To address your soundness concern, we have revised Section III. A to explicitly describe: (1) what we analyze in C# scripts and scene/prefab assets; (2) how we extract object–script bindings and UnityEvent linkages; and (3) how the dynamic process captures `Instantiate()`/`AddComponent()` and validates interaction logic via probes. We excerpt the major revisions as follows.

In Section III.A (Project Collection and Analysis):
"Static pass (S1–S3).

- *S1. Scene/prefab scan***.** We parse scene hierarchies and prefabs to enumerate GameObjects with attached components, and prune *non-interactable* infrastructure (e.g., static walls) that are not connected to core interactions.
- *S2. Script binding extraction.* We inspect MonoBehaviour-based scripts and their serialized fields to recover object–script bindings (e.g., references to targets).
- *S3. Interaction pattern identification.* We analyze interaction-related APIs and callback signatures to fingerprint typical behaviors.

The interaction distribution will be detailed in Section IV-A.

Dynamic pass (D1–D3).
Static inspection cannot observe runtime object creation, late component attachment, or goal-oriented task logic. To complement this effect, we conduct a lightweight but human-guided runtime analysis guided by gameplay flows and task objectives:

- *D1. Runtime state observation.* At each frame t, we record snapshots of active GameObjects G_t and their component sets C_t. Comparing consecutive states, we identify dynamically instantiated objects and runtime-added components.
- *D2. Interaction flow tracing.* Following execution evidence such as console outputs, function call stacks, and Unity event dispatches during gameplay, we trace when and how scripts are actually triggered (e.g., input events, collision handlers, and task-specific event chains).

- *D3. Heuristic validation of task logic.* Four experienced engineers heuristically reconstruct event sequences observed during gameplay to approximate these flows and validate whether each interaction contributes essentially to task progression or it is only incidental."

For more details, please refer to the highlighted version of the revised paper.

- In Section III.B, the paper discusses the Model Abstraction process. However, the abstraction procedure appears to be largely manual and covers only a limited set of VR object types. In practice, VR applications can include a wide range of objects, such as those involving physical manipulation, trigger-based interactions, and environmental or scene-level components. The paper does not clearly explain how these categories were defined or selected. If the abstraction process is manual, the authors should explicitly state who was involved and outline the steps followed during model construction. Greater transparency here would improve reproducibility and clarify the scalability of the approach.
-

Response: Thanks for your comment.

Notably, the model abstraction process is human-in-the-loop due to the complexity and diversity of VR interactions. During project analysis, test engineers summarize heuristic experiences and rules, merging static and dynamic findings into abstraction models. Common VR interaction behaviors are classified into abstract actions based on OOP principles, and objects are annotated with interactable features such as Grabbable, Triggerable, and Transformable. These abstractions seed the Entity Interface Layer and Action Class Layer in the EAT framework. Then, surface callable functions and UnityEvents populate task and exploration nodes in the PFSM behavior space.

To improve transparency and reproducibility, we explicitly note that trained test engineers conducted the manual analysis following the two-pass procedure as described in Section III-A. New interaction patterns can be matched to existing types or newly defined via new Action interfaces, and execution remains fully automated once the model is constructed.

We excerpt the revised parts as follows:

In Section III-B (Model Abstraction):
"**Heuristic Model Abstraction.** We summarize the heuristic experiences and rules extracted from the project analysis in order to test the framework's effectiveness and get a preliminary model abstraction. Specifically, during the project analysis process, we merge static and dynamic findings into abstraction models heuristically and classify common VR interaction behaviors into abstract actions based on OOP principles, as shown in Part B in Fig. 1. When testing a real-world project, engineers can absolutely use the model we had constructed, while also experiencing core gameplay and performing the two-pass manual analysis as described in Section III-A if they have customized testing demands. New interaction patterns are either matched with existing types or newly defined via new Action interfaces automatically. While execution is fully automated, abstraction still requires human-in-the-loop analysis due to the complexity of VR interactions and the limited domain-specific knowledge of existing tools..

Objects are annotated with interactable features (e.g., Grabbable, Triggerable, and Transformable); interactions are lifted into abstract actions. This consolidated model (i) seeds the *Entity Interface Layer* in EAT by proposing interface candidates (e.g., IGrabbable and ITriggerable), (ii) provides action-level semantics to the Action Class Layer, and (iii) surfaces callable functions and UnityEvents to populate the task. (Section III-D)."

- The use of NavMesh for path exploration in VR environments is a compelling and well-justified choice, and I appreciate its integration into the testing framework. However, the construction process for the probabilistic finite state machine (PFSM) model is not clearly described. While the probabilistic modeling approach seems appropriate for the shooting application—where stochastic behaviors can naturally reflect different application states—the paper does not explain how this modeling would extend to other types of VR applications (e.g., puzzle-based experiences or educational simulations). If the PFSM is constructed in an application-specific manner, this raises concerns about the generalizability of the approach. More discussion is needed to clarify whether the model can be adapted across diverse VR domains or if it requires significant manual tailoring per application.

Response: Thanks for your comment.

PFSM is suitable for diverse game types. For another example, puzzle tasks can be modeled using object triggers or movement tasks. These *game-independent* abstractions require *no additional configuration* from test engineers. In the submitted paper, we selected the shooting game as a representative example, though our framework essentially considered *diverse types* of VR projects, including simulation, adventure, puzzle games, strategy board games, medical care apps, etc.

In the revised paper, we have supplemented these additional details.

In Section VI-E (Generality of VRExplorer):
"Our approach systematically unifies these diverse interactions by decomposing VR objects into a finite set of interactable features, which are then represented as abstract actions. Based on OOP principles, the Entity Layer and Action Layer encapsulate input modalities and target objects separately, effectively treating them as reusable and generic components. Within this feature–action hierarchy, abstract actions are further integrated into PFSM as semantic nodes and transitions, allowing frequent and domain-specific behaviors to be expressed uniformly. This design ensures generality across projects while preserving extensibility for new interaction manners.

For those not mentioned in Section III, it only requires modeling a simple composition of task nodes and an additional configuration, without modifying the core logic. This OOP-based design allows the framework to scale naturally to a wide spectrum of VR interactions. For example, *GUI Interactions* (e.g., inputting texts and pressing a 2D button) can be represented by customized tasks composed of *Trigger Action* nodes, which provide event lists and function call chains for simulating different GUI interactions. Similarly, *Hand Gesture*-based interactions can be supported by treating gestures as input-trigger components in the Entity Layer, while a dedicated *Gesture Action* class in the Action Layer encapsulates concrete gestures such as swipe,

pinch, and point. These gestures can then be mapped to task transitions in the PFSM, e.g., "swipe-to-turn-the-page" or "pinch-to-zoom."

In summary, the EAT framework is inherently agnostic to input types. In other words, each project only needs to define its input actions and associated targets, after which abstraction and task composition are handled systematically through the generic OOP framework."

- The paper states that a constructed dataset was used to perform a preliminary analysis, from which both the object and action abstractions were derived. It appears that this same dataset is also used for evaluating the proposed framework and conducting the comparative analysis with the state-of-the-art approach, VRGuide. This raises a significant risk of data leakage, where insights gained during model construction may inadvertently influence evaluation outcomes. Furthermore, the project selection criteria are insufficiently detailed. It is unclear how diverse the evaluation set is across different types of VR applications**.** Based on the descriptions, I suspect that the evaluation projects are predominantly shooting-based or have similar event transition steps, which closely align with the modeling assumptions and structure of the proposed approach. This alignment could unintentionally bias the results, potentially overstating the performance advantage over VRGuide. A clearer justification of dataset diversity and separation between training/modeling and evaluation phases is needed to ensure the validity and fairness of the empirical analysis.

Response: We thank the reviewer for raising this important concern.

We would like to clarify that our approach is not learning-based and therefore does not involve training or finetuning on a dataset. Instead, the dataset analysis serves only to identify common VR interaction patterns and to guide the definition of heuristic rules. As a result, our method has no risk of "*data leakage*" unlike conventional learning-based approaches since no training/testing split or parameter tuning is performed.

To further address the concern about dataset diversity and representativeness, we have also added a new subsection in Section IV-A. As shown in Table IV and Figures 5-6, our dataset includes 102 projects spanning a wide range of sizes (from very small projects with only a few scripts to very large ones with thousands of scripts and hundreds of scenes) and interaction types (from prevalent ones such as grabbing and teleportation to infrequent ones such as inventory and dialogue). These interactions cover all major categories summarized in prior VR interaction taxonomies, ensuring that the dataset is not biased toward a single genre such as shooting games.

We excerpted the revised parts as follows:

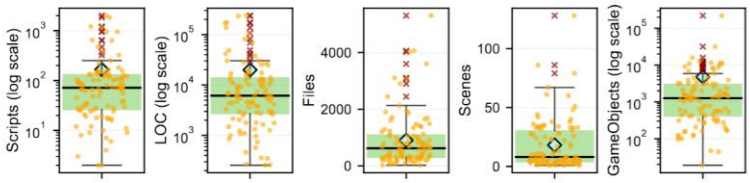

Fig. 5: Boxplots of scripts, LOC, files, and scenes.

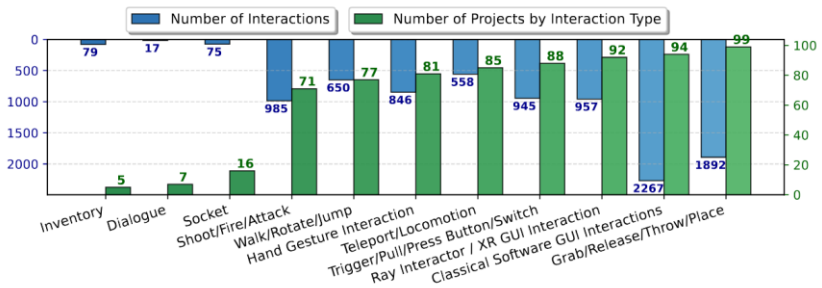

Fig. 6: Distribution of Interactions in Dataset.

TABLE IV: Statistical Metrics of Collected 102 Projects.

Metric	Mean	Variance	Min	Q1	Median	Q3	Max
Scripts	166.45	110,953.52	2	26.25	71.50	130.25	2,004
LOC	19,910.85	1,851,165,810.13	256	2,707.25	6,129.00	13,762.75	237,725
Files	902.07	902,718.22	21	301.25	620.50	1,090.00	5,303
Scenes	18.12	455.06	1	4.00	8.00	30.25	128

In Section IV-A (Implementation):
"**Project Collection and Analysis.** We implemented a crawler to collect Unity-based VR GitHub projects with keywords like "VR", "AR", "unity", and "xr". From 971 initially filtered projects, we manually performed quality checks (e.g., compilation errors and version conflicts), consequently retaining 102 high-quality projects.

We then analyze the structural and interaction characteristics of all projects in the dataset to assess their representativeness and complexity. Table IV shows varied project sizes. While most projects are relatively small (medians containing 71.5 scripts, 6,129 LOC, 620 files, and 8 scenes), a few of them with large sizes markedly raise medians. Fig. 5 plots their distribution. Fig. 6 indicates that the dataset includes both prevalent interactions (e.g., grabbing, GUI operations, and teleportation) and infrequent ones (e.g., inventory and dialogue). (e.g., inventory and dialogue). Together, these interactions cover all major interaction tasks identified in [28], thereby highlighting the constructed dataset's representativeness and diversity.

We then perform the two-pass analysis (Section III-A), summarizing heuristic rules and experiences as drafted documentation for subsequent modeling."

[28] J. Hertel, S. Karaosmanoglu, S. Schmidt, J. Br¨aker,M. Semmann, and F. Steinicke, "A taxonomy of interaction techniques for immersive augmented reality based on an iterative literature review," in 2021 IEEE International Symposium on Mixed and Augmented Reality (ISMAR),2021, pp. 431–440.

In summary, (1) our method involves no training and hence is not subject to data leakage risks, and (2) our evaluation set is both structurally diverse and interactionally comprehensive, as explicitly analyzed in Section IV-A. This consideration ensures that the empirical analysis is fair and representative.

- As part of RQ3, the paper reports that the approach identified three real-world bugs. However, it appears that these bugs were not confirmed by the original developers of the

applications. This raises a question about how the authors validated that these were indeed legitimate bugs rather than false positives or expected behaviors. Clarifying the criteria used to label these findings as bugs—whether through manual inspection, heuristics, or expert validation—would strengthen the credibility of this result and help assess the practical impact of the proposed approach.

Response: Thanks for your comment.

In the revised version, we have explicitly clarified our validation process and the evidence supporting the legitimacy of these bugs. We added a new subpart, Section V-C (Bug Detection in VR Projects: **Detected Bugs Validity Confirmation**), to give more details about the criteria used to label these findings as bugs through Unity console logs, runtime exception traces, and script-trigger chains.

We excerpt the revised parts as follows:

In Section V-C (Bug Detection in VR Projects):
"**Detected Bugs Validity Confirmation.** To further confirm the validity of detected bugs, we validated all reported issues through Unity console logs, runtime exception traces, and script-trigger chains. Among them, the bug in *UnityCityView* has been confirmed and fixed by its developers (evidenced by subsequent commits). For the two bugs in *EscapeGameVR*, we performed root-cause analyses: the functional bug is an *Unassigned Reference Exception*, consistent with CWE-395 [35] and CodeQL's rule on improper initialization [36], while the non-functional bug is caused by a missing ArrowPrefab resource, which is a common Unity issue documented in the official Prefab guidelines [37]. We also reported these issues to the original developers of EscapeGameVR, but no response was received till now."

[37] Unity Technologies, "BrokenPrefabAsset," Unity Documentation, https://docs.unity3d.com/6000.1/Documentation/ScriptReference/BrokenPrefabAsset.html, 2025.

Novelty

- Overall, I really like the idea of using a modeling approach and path exploration based on NavMesh—it's a promising direction. However, I found that much of the current approach appears to be manual and potentially tailored to specific application types, such as shooting games or other similar event-driven VR experiences. To improve its generality and applicability, I believe the approach should incorporate more automated modeling capabilities, driven by static code and GameObject analysis. This would allow it to support a broader range of VR applications and make the method more scalable and adaptable to diverse interaction contexts.

Response: We thank the reviewer for the positive feedback and comments regarding the generality and automation of our approach.

We would like to clarify that our method is not fully manual. Our framework employs a semi-automated abstraction process: once VR interactions are identified and annotated, they are mapped into the Entity and Action layers, and subsequent task execution is fully automated via the PFSM-based exploration.

Manual analysis is only required for human-in-the-loop abstraction, i.e., defining new Action interfaces or annotating interactable object features (e.g., Grabbable, Triggerable, and Transformable) when novel interaction patterns are encountered. This step is necessary because current tools lack enough domain-specific knowledge to automatically reason the semantics of diverse VR interactions and scene-specific behaviors. This work can be further automated by tools (e.g., LLMs) in the future.

All other aspects, including statistical analysis, task execution, action sequencing, and scene exploration, are **completely automated**. Moreover, test engineers can **reuse previously constructed models** for new projects or extend them with minimal effort by performing the two-pass manual analysis described in Section IV-A if customized testing is required.

Presentation
- Overall, the paper is easy to read and understand.

Verifiability
- Artifacts are available. However, the readme file seems empty. The readme file should include steps to reproduce the artifact.

Questions for authors' response
Q1. Based on the Approach and Threats to Validity sections, it appears that the analysis and modeling process is primarily manual. Could you please clarify the specific steps involved in this manual analysis?

Response: Thanks for your positive comment.

Regarding your concern about the analysis and modeling process, our framework employs a *semi-automated* approach. The specific steps involved in our whole process are listed as follows.

Static pass: *Semi-Automated* (Specific steps: S1. Scene/prefab scan (***Manual***), S2. Script binding extraction (*Semi-Automated*) and S3. Interaction pattern identification (*Semi-Automated*))
Dynamic pass: *Semi-Automated* (Specific steps: D1. Runtime state observation, D2. Interaction flow tracing and D3. Heuristic validation of task logic)

Heuristic Model Abstraction: ***Manual***

VRExplorer Testing: (fully-automated)
- Scene Interaction: Automated
- Scene Analysis: Automated
- PFSM decision: Automated

- Task execution: Automated
- Scene Exploration: Automated
- Autonomous Event Invocation: Automated
- Task Generation: Automated
- Bug Detection: Automated

Correspondingly, we have also made a minor revision on the title of this paper to reflect the revision.

Q2. Do you believe your current modeling process can generalize to a wide range of VR applications beyond the use case(s) demonstrated in the paper? For instance, how would it handle diverse interaction paradigms such as gesture-based input, gaze-based selection, or multi-user collaboration environments?

Response: Thanks for your question.

Yes, our modeling process is designed for the scalability of diverse VR projects. The Action layer abstracts user interactions by decoupling input modalities from target objects. The EAT framework itself is *agnostic* to specific input types since each project only needs to provide input actions and corresponding targets. The diverse input types [14][15] can be flexibly added as new input types into our EAT framework.

[14] J. Wentzel, M. Lakier, J. Hartmann, F. Shazib, G. Casiez, and D. Vogel, "A Comparison of Virtual Reality Menu Archetypes: Raycasting, Direct Input, and Marking Menus," IEEE Transactions on Visualization and Computer Graphics, vol. 31, no. 9, pp. 4868–4882, 2025.
[15] T. Wan, Y. Wei, R. Shi, J. Shen, P. O. Kristensson, K. Atkinson, and H.-N. Liang, "Design and evaluation of controller-based raycasting methods for efficient alphanumeric and special character entry in virtual reality," IEEE Transactions on Visualization and Computer Graphics, vol. 30, no. 9, pp. 6493–6506, 2024.

In the revised paper, we have supplemented these additional details.

In Section VI-E (Generality of VRExplorer):
"Our approach systematically unifies these diverse interactions by decomposing VR objects into a finite set of interactable features, which are then represented as abstract actions. Based on OOP principles, the Entity Layer and Action Layer encapsulate input modalities and target objects separately, effectively treating them as reusable and generic components. Within this feature–action hierarchy, abstract actions are further integrated into PFSM as semantic nodes and transitions, allowing frequent and domain-specific behaviors to be expressed uniformly. This design ensures generality across projects while preserving extensibility for new interaction manners.

For those not mentioned in Section III, it only requires modeling a simple composition of task nodes and an additional configuration, without modifying the core logic. This OOP-based design allows the framework to scale naturally to a wide spectrum of VR interactions. For example, *GUI Interactions* (e.g., inputting texts and pressing a 2D button) can be represented by

customized tasks composed of *Trigger Action* nodes, which provide event lists and function call chains for simulating different GUI interactions. Similarly, *Hand Gesture*-based interactions can be supported by treating gestures as input-trigger components in the Entity Layer, while a dedicated *Gesture Action* class in the Action Layer encapsulates concrete gestures such as swipe, pinch, and point. These gestures can then be mapped to task transitions in the PFSM, e.g., "swipe-to-turn-the-page" or "pinch-to-zoom."

Point-by-Point Response to Reviewer C

Paper summary

The study proposes VRExplorer, an approach to test the diverse interactions in a virtual environment. VRExplorere is based on EAT framework (Entity, Action, Task), a testing tool, to interact and explore the virtual application. The study outperformed the current state-of-the-art model on 9 VR projects, improving performance by 72.8% and 46.9% in executable lines of code coverage and method coverage, respectively.

Strengths
+ Useful Testing Tool.
+ Enhanced Interaction Coverage.

Weaknesses
- Confusing subsection heading.
- Missing guidelines to replicate the study.
- Minor Presentation issues.

Response: Thanks a lot for your positive comments on this paper. We also appreciate your constructive comments, which help us greatly improve the paper by conducting a comprehensive major revision. We present the detailed responses to your concerns in a point-by-point manner as follows.

Detailed comments for authors

The paper presents an approach based on the EAT (Entity, Action, Task) framework, showcasing results that outperform the baseline model. The proposed method has practical value and can be utilized by developers and test engineers to evaluate their applications and improve overall software quality.

The paper presents a new approach compared to existing methods. While prior approaches rely on a single type of interaction (e.g., mouse click), the proposed method incorporates multiple interaction types such as move, grab, and trigger. This broader action coverage contributes to its improved performance over existing work.

The paper provides a replication package, which enhances the transparency of the work. However, there is no procedure on how we can replicate the study; the readme file only includes the title of the study.

There are some minor presentation issues in the manuscript. The issues include discrepancies in the subheading:
In Section Overview (D. VRExplorer Testing), the title of paragraph two is labeled as "The Entity Interface Layer", whereas the same component is referred to as "Implementation Interface" in Fig. 1 and again in Section D (VRExplorer Testing), first paragraph. This inconsistency in naming breaks the coherence of the presentation and may confuse the reader. It would be helpful to standardize the terminology throughout the paper for clarity.

Response: Thank you for your positive comments on our research work.

In the Experimental Result section B (Ablation Study), Ablation Experiment Configuration, second paragraph: in the third sentence

"For example, we obtain VRExplorer w/o T by removing the Triggerable module from the Entity layer's interface Triggerable, the Action layer's class TriggerAction, all tasks involve triggering in the Task layer, and the corresponding Mono C# scripts XRTriggerable.cs.
"

If "w/o" stands for "without," as clarified in the following sentence, it would improve readability to introduce that abbreviation earlier. Since "w/o" is used without prior explanation, adding "without (w/o)" when it first appears would help readers unfamiliar with the abbreviation.

Response: Thanks for your comment. We have clarified "w/o" by introducing it as "without (w/o)" when first mentioned, improving readability.

Minor issues

The paper is easy to read and follow, though there are a few minor areas that could be improved.
The layout of Fig. 1 (Overview) can be improved by aligning it with the order of the sections (e.g., A, B, C, D). For instance, in the second column, it would enhance clarity if Section C were placed before Section D.

Response: Thanks for your comment. We have adjusted Fig. 1 to better align its layout. The revised version of Fig. 1 is shown as follows:

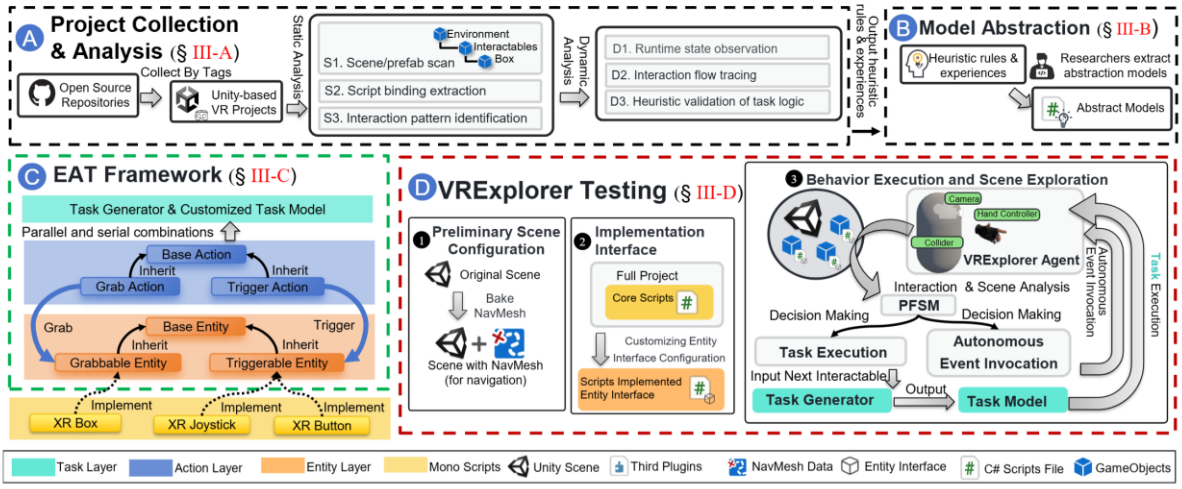

Fig. 1: Overview of *VRExplorer*

In Table VII, it is stated that underlining indicates the highest performance. However, the table also uses a gray background to highlight certain entries (e.g., VRExplorer w/o *T*, VRExplorer w/o *Tf*), which visually draws more attention than the underlining. To improve clarity, it would be better to use bold text in addition to underlining to highlight the best-performing results.

Response: Thanks for your comment.

In the revised version, we now emphasize the best results using boldface combined with underlining, while keeping the gray background to indicate ablation settings. We hope this revision improves the clarity and readability of the table.

Questions for authors' response
- Will the selected Core code be useful for many other types of VR applications? If not, will it introduce additional workloads for the developers?

Response: Thanks for your question.

Yes. We proposed *Core Code* as a *heuristic criterion* to select only the modules implementing core VR interaction functionality, while excluding tool libraries, third-party code, or other non-interaction scripts. This consideration ensures that testing and code coverage measurement focus on key codes that actually affect VR interactions, while also allowing flexible customization for different VR projects.

Thus, this design is applicable to other types of VR applications (even other Unity-based or non-Unity-based games). In the revised paper, we have clarified this design.

We excerpt the revised parts as follows:

In Section IV-A (Implementation):

“For each project, we analyze the interactable objects in scenes and all the C\# scripts referenced by the objects, and select the codes corresponding to the core VR interaction functionality, referred to as *Core Code*.

Tool libraries, third-party code, and other non-interaction scripts are excluded. This selection serves two purposes: (i) evaluation of testing performance via Assembly-Definition files, and (ii) flexible customization and configuration for different VR projects.

Core Code is a *heuristic criterion* useful for any type of VR applications (even Unity-based or non-Unity-based).”