IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.

Unveiling Non-Reproducibility in LiDAR-Inertial Odometry

Hongqian Huang, *Graduate Student Member, IEEE*, Meng Zhang, *Senior Member, IEEE*, Jianchen Hu, *Member, IEEE*, and Xiaohong Guan, *Life Fellow, IEEE*

***Abstract*—This letter presents empirical research on the non-reproducibility of light detection and ranging sensor (LiDAR)-inertial odometry (LIO) systems. Although the LIO community has made commendable efforts toward reproducible localization accuracy, *noteworthy* non-reproducibility remains, thus hindering a fair evaluation of method effectiveness. To better understand such non-reproducibility, we first define non-reproducibility and introduce a quantitative criterion to identify noteworthy non-reproducibility. We then propose five significant non-deterministic implementations that are included in state-of-the-art LIO systems and present solutions for modifying these non-deterministic implementations into deterministic ones. A general procedure is also introduced to identify and pinpoint non-deterministic implementations, regardless of whether they are covered in this letter. Extensive experiments demonstrate that the non-deterministic implementations are the major or potentially sole causes of non-reproducibility under constant experimental conditions. Additionally, the non-reproducibility is noteworthy in datasets obtained from low-vertical-resolution LiDARs or recorded in geometrically degenerate scenes.**

***Index Terms*—LiDAR-inertial odometry, reproducibility, deterministic implementation.**

I. Introduction

Reproducibility is essential for scientific research, as it enables credible evaluation of method effectiveness. To this end, LIO researchers attempt to ensure reproducibility (specifically, reproducible localization accuracy) by standardizing runtime environments through containerization [1], [2], [3] and sharing code as well as algorithm parameter configurations [4], [5], [6], [7], [8]. However, these efforts are not always sufficient to ensure reproducibility (see Fig. 1).

Empirically, non-deterministic implementations have been observed to contribute to the emergence of non-reproducibility. In machine learning, non-deterministic implementations of randomized algorithms (such as random initialization of model parameters, dropout [9], and random data augmentation [10]) are deliberately employed to improve model generalization and prevent overfitting during training. A common approach to transforming such non-deterministic implementations to deterministic ones is to fix the seed of the random number generator.

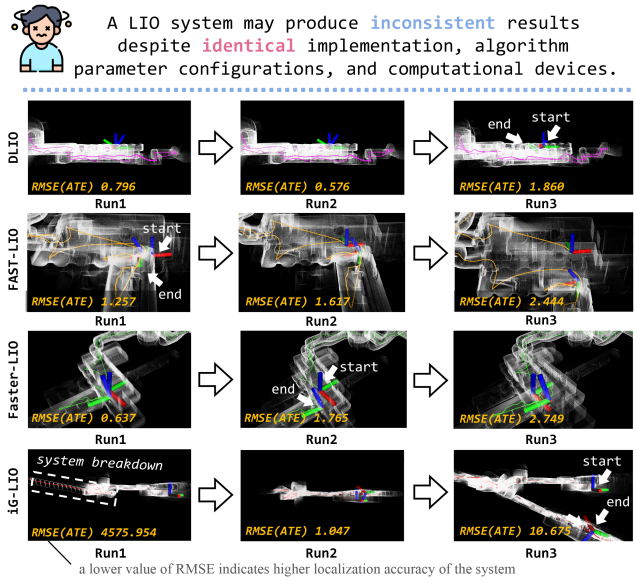


Fig. 1. An illustration of non-reproducible LIO systems including DLIO [5], FAST-LIO2 [6], Faster-LIO [7], and iG-LIO [8]. The experiments were conducted in the Sewerage sequence [11] under **identical** experimental conditions across all runs. Despite these controlled conditions, the systems demonstrate significant inconsistency in root mean square error (RMSE) of absolute translation errors (ATE), with variations exceeding 1 m between different runs. Note that RMSE reflects the overall deviation across the entire trajectory. Once an estimated pose significantly deviates from the ground truth and the resulting misalignment propagates over time, the overall RMSE can reach several hundred or even thousands of meters. Such large values generally indicate a *system breakdown*.

However, this approach is generally inapplicable to LIO systems, because they rarely involve randomized algorithms. Therefore, identifying LIO-specific sources of non-determinism becomes necessary. Moreover, if these non-deterministic implementations do not contribute positively to the system's performance or robustness, methods should be developed to convert them into deterministic counterparts. Furthermore, as non-reproducibility may result from the combined effects of multiple non-deterministic implementations, we investigate the extent to which each non-deterministic implementation contributes to the non-reproducibility.

To the authors' knowledge, this work provides the first empirical evidence that non-deterministic implementations in LIO systems can cause noteworthy non-reproducibility. The main contributions are as follows:

Received 7 August 2025; accepted 18 November 2025. Date of publication 24 November 2025; date of current version 5 December 2025. This article was recommended for publication by Associate Editor M. Camurri and Editor J. Civera upon evaluation of the reviewers' comments. *(Corresponding author: Meng Zhang.)*

Hongqian Huang and Meng Zhang are with the School of Cyber Science and Engineering, Xi'an Jiaotong University, Xi'an 710049, China (e-mail: hong877381@gmail.com; mengzhang2009@xjtu.edu.cn).

Jianchen Hu and Xiaohong Guan are with the School of Automation Science and Engineering, Xi'an Jiaotong University, Xi'an 710049, China (e-mail: horace89@xjtu.edu.cn; xhguan@xjtu.edu.cn).

Digital Object Identifier 10.1109/LRA.2025.3636030

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.

- Five instances of non-deterministic implementations are proposed, thus offering a practical checklist for the preliminary identification of sources contributing to non-reproducibility. A general procedure is also introduced to support the discovery of both listed and unlisted non-deterministic implementations.
- Extensive experiments are conducted to demonstrate that non-deterministic implementations are the dominant and potentially sole causes of non-reproducibility under constant experimental conditions. Such non-reproducibility becomes noteworthy when systems are evaluated on datasets collected with low-vertical-resolution LiDARs (e.g., 2°) or captured in geometrically degenerate environments (e.g., flat fields). These experimental findings underscore the risk of drawing unreliable or false-positive conclusions when evaluating under such datasets.
- Ablation studies are conducted on LIO systems incorporating multiple non-deterministic implementations to demonstrate that each of the five non-deterministic implementations independently leads to noteworthy non-reproducibility. Among the five implementations, the hardware-dependent one is further validated that the resulting non-reproducibility stems from variations in the addition order during matrix multiplication. These findings open up new opportunities for enhancing the accuracy and robustness of LIO systems.

The remainder of this letter is organized as follows. Section II defines reproducibility in the context of LIO. Section III presents the methodology for identifying implementations that cause the non-reproducibility and details five non-deterministic implementations. Section IV details experimental setups. Section V presents qualitative and quantitative analyses of the non-reproducibility. Section VI concludes the letter.

II. Reproducibility

This section defines the notion of reproducibility and explains why non-reproducibility poses a challenge to fair method evaluation.

Definition (Reproducibility): Our study adapts the reproducibility definitions proposed in [12] to the context of LIO. A LIO system and its results (e.g., accuracy) are considered reproducible if repeated experiments yield consistent results when experimental conditions (including dependency library versions, implementations, algorithm parameter configurations, evaluation methodology, and computation devices) are held constant. Otherwise, the system and its results are deemed non-reproducible. Throughout this letter, the terms *reproducible/non-reproducible systems* and *systems with reproducibility/non-reproducibility* are used interchangeably.

Definition (Noteworthy non-reproducibility): A LIO system is described as exhibiting noteworthy non-reproducibility if the range of accuracy exceeds $1\,\mathrm{m}$. Specifically, the range is defined as the difference between the maximum and minimum RMSE values (used as the measure of accuracy) from repeated experiments.

Remark 1: The $1\,\mathrm{m}$ threshold is empirically determined from multiple benchmark datasets [4], [11], [13], [14], [15], where variations beyond this level consistently correlate with unstable localization. Additionally, such variations often indicate possible software defects in implementation. Note that this threshold serves as a necessary condition for identifying noteworthy non-reproducibility in our experimental setup, but alternative thresholds may be more appropriate for different datasets or application scenarios.

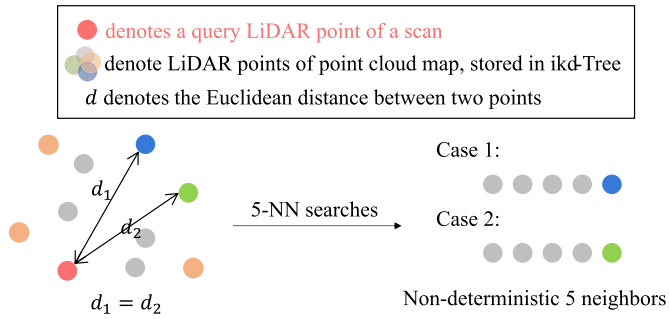


Fig. 2. An illustration of non-deterministic nearest neighbors in ikd-Tree based on the original implementation [6].

Remark 2: While the term range is a standard statistical term, it may introduce ambiguity in the context of LIO systems, as it can be misinterpreted as a spatial metric. To avoid this confusion, we instead use the term *accuracy variation* to describe the extent of non-reproducibility.

Suppose we aim to evaluate the effectiveness of a proposed method. We rank accuracy from multiple state-of-the-art systems, the system with the method applied, and the same system without the method. For systems that yield inconsistent but statistically similar numerical results (i.e., with accuracy variations below a predefined threshold), we use the mean value for comparison. Furthermore, a higher rank of the method-enhanced system indicates greater effectiveness of the proposed method.

Unfortunately, the effectiveness evaluation may become unfair when LIO systems exhibit noteworthy non-reproducibility. Specifically, in such cases, neither the mean value nor the cherry-picked optimal result can fairly or accurately reflect the true performance of systems. Therefore, mean/optimal selection should not be used for comparison. Instead, primary efforts should focus on eliminating such non-reproducibility.

III. Methodology

A. Five Non-Deterministic Implementations

This section presents five identified non-deterministic implementations. The methods for transforming these non-deterministic implementations into deterministic ones are also presented, and the corresponding code is available at `https://liobench.github.io/LIOBench`.

1) Non-Deterministic Nearest Neighbors: The double-thread rebuilding method used in hierarchical data structures such as ikd-Tree [6] can yield non-deterministic tree hierarchies, which consequently lead to non-deterministic k-nearest neighbors (see Fig. 2), even when the data structures store the same point clouds.

Measures: To ensure deterministic nearest neighbors (NN), the original NN search condition can be modified from a strict ($<$) to a non-strict inequality ($\leq$). This modification enables the search to incorporate map points that are located exactly at a current maximum allowed distance from query points, where the current maximum allowed distance is defined as the largest distance between the query point and any of the currently selected k nearest neighbors. The modified search condition, when

integrated with a tie-breaking strategy (selecting LiDAR points with smaller x-coordinates when encountering equidistant map points), largely ensures deterministic nearest neighbor selection.

Affected systems: Systems such as FAST-LIO2, COIN-LIO, LOG-LIO [16] that employ data structure (like ikd-Tree) with non-deterministic tree hierarchies.

2) Hardware-Dependent Implementation: Computations on modern heterogeneous processors may yield two numerical outcomes, depending on whether the computation is executed on performance cores (P-cores) or efficiency cores (E-cores). Since thread schedulers assign threads to different cores in a non-deterministic manner unless CPU affinity is explicitly specified, the numerical results are non-deterministic. Although a comprehensive investigation into the root causes of these hardware-dependent results would offer deeper insights, such an analysis is beyond the scope of our work.

Measures: Enforce execution on specific cores (e.g., P-cores) through CPU affinity settings.

Affected systems: We observe such non-determinism in COIN-LIO, FAST-LIO2, and Faster-LIO. Specifically, the `Eigen`-based computation of $\mathbf{H}^\top\mathbf{H}$ (where $\mathbf{H} \in \mathbb{R}^{j\times 12}$) may yield different 12×12 matrices depending on whether the computation is executed on P-cores or E-cores. Here, $\mathbf{H} = [\mathbf{H}_1^\top, \mathbf{H}_2^\top, \ldots, \mathbf{H}_j^\top]^\top$, where each $\mathbf{H}_j \in \mathbb{R}^{1\times 12}$ is the Jacobian matrix of the j-th measurement function with respect to the error state, and j represents the measurement dimension. Note that small matrix multiplications do not exhibit this non-determinism, and determining the lower bound of matrix dimensions for the non-deterministic numerical results also falls outside the scope of our work.

3) Thread-Unsafe Implementation: Thread-unsafe modifications to the same elements without synchronization or using inherently thread-unsafe structures (e.g., containers like `std::vector<bool>`[1]) may lead to unpredictable data races and incorrect behaviors.

Measures: Two approaches for ensuring thread safety are (1) using thread-safe data structures explicitly designed for concurrent operations, such as `std::vector<int>` and (2) employing synchronization primitives (e.g., mutexes, locks) or atomic operations to prevent data races.

Affected systems: The thread-unsafe implementation can be observed in systems COIN-LIO and Faster-LIO, where multiple threads concurrently modify `std::vector<bool>`, resulting in unintended overwrites and inconsistent logic execution. Specifically, in Faster-LIO, LiDAR points that should have been excluded from residual computation may incorrectly be included due to these overwrites.

4) Parallel Floating-Point Reduction: Floating-point reduction operations, such as matrix addition, are widely used in multi-threaded environments. These operations can exhibit non-deterministic behavior due to the combined effects of the non-deterministic execution order of reduction operations (determined by thread scheduling) and the non-associativity of floating-point arithmetic [18] (see Fig. 3).

Measures: To obtain deterministic results, one approach is to store the floating-point objects (e.g., numbers, matrices) to be reduced in an array within a multi-threaded environment, and then apply the reduction operator over this array in a single-threaded environment. An overview of the implementation of this approach is provided in Listing 1.

[1]As noted in `https://en.cppreference.com/w/cpp/container/vector_bool`, `std::vector<bool>` is typically implemented as a bitset-like structure in which multiple boolean values share the same underlying storage.

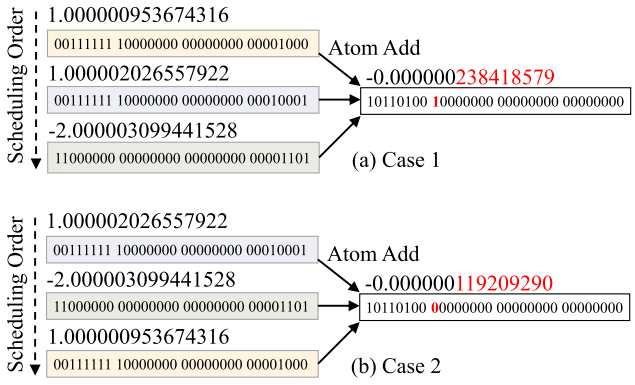


Fig. 3. An illustration of the non-associativity of floating-point addition. The boxed values show the 32-bit IEEE 754 [17] single-precision floating-point binary representations as stored in memory, while the decimal numbers above each box provide their corresponding 15-decimal-place interpretations. For example, the binary sequence 00111111 10000000 00000000 00001000 stored in memory is decoded as the decimal value 1.000000953674316 when represented to 15 decimal places.

```
- double nonDeterministicSum(vector<double> &input) {
+ double deterministicSum(vector<double> &input) {
    double result = 0.0;
+ vector<double> results(input.size(), 0.0);
- #pragma omp parallel for reduction(+:result)
+ #pragma omp parallel for
    for (uint i = 0; i < input.size(); ++i) {
// func is a function with signature: double func(double)
-     result += func(input[i]);
+     results[i] = func(input[i]);
    }
+ for (uint i = 0; i < input.size()) {
+     result += results[i];
+ }
    return result;
}
```

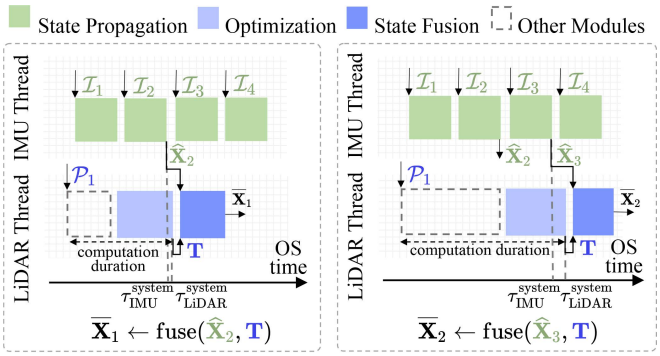
Listing 1. Simplified code modifications marked (+ additions / − deletions) for deterministic floating-point reduction in multi-threaded environments, relative to the non-deterministic implementations [5]. Note that the code snippet contains only the algorithm's critical logic, with auxiliary code omitted for clarity.

Affected systems: This type of non-determinism can be observed in systems such as DLIO and iG-LIO. Specifically, iG-LIO employs `tbb::parallel_reduce` to parallelize the accumulation of Hessian matrices, while `DLIO` uses `OpenMP reduction` to parallelize the accumulation of Mahalanobis distance errors and spaciousness metrics [19].

5) Non-Deterministic State Fusion: State-fusion systems that synchronize states from different sensors using operating system (OS)-generated timestamps may exhibit non-deterministic results. This issue arises because OS-level timestamps (e.g., time of message publication or processing) are subject to variability caused by thread scheduling and computation latency. Consequently, fusion modules that select sensor states based on these timestamps may produce inconsistent results across runs.

Measures: Employ sensor-generated acquisition timestamps, which are intrinsic to the sensor data and unaffected by the latency.

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.



(a) Case 1 (b) Case 2

Fig. 4. An illustration of the non-deterministic selection of the propagated state $\widehat{\mathbf{X}}$ (e.g., $\widehat{\mathbf{X}}_2$ or $\widehat{\mathbf{X}}_3$) for sensor fusion. The figures present the timeline for processing an inertial measurement unit's (IMU) data $\mathcal{I}_1$, $\mathcal{I}_2$, $\mathcal{I}_3$, $\mathcal{I}_4$ and point cloud $\mathcal{P}_1$ in parallel within DLIO. The optimization module (light blue block) outputs the transformation matrix T and the state propagation module (green block) outputs the propagated state $\widehat{\mathbf{X}}$. The timestamp $\tau^{\text{system}}_{\text{LiDAR}}$ and $\tau^{\text{system}}_{\text{IMU}}$ are defined as the output times of T and $\widehat{\mathbf{X}}$, respectively, both measured by the operating system (OS) clock. In the sensor fusion module (dark blue block), T fuses with $\widehat{\mathbf{X}}$ whose $\tau^{\text{system}}_{\text{IMU}}$ is closest to $\tau^{\text{system}}_{\text{LiDAR}}$.

Affected systems: For example, the non-reproducibility of DLIO mainly stems from the non-deterministic selection of the propagated state $\widehat{\mathbf{X}}$ for sensor fusion (see Fig. 4). This non-deterministic selection arises from the selection rule that the state $\widehat{\mathbf{X}}$ with $\tau^{\text{system}}_{\text{IMU}}$ closest to $\tau^{\text{system}}_{\text{LiDAR}}$ is selected for sensor fusion, where $\tau^{\text{system}}_{\text{LiDAR}}$ represents the operating system (OS) time for the output of the transformation matrix $\mathbf{T}$ and $\tau^{\text{system}}_{\text{IMU}}$ is the OS time for the propagated state $\widehat{\mathbf{X}}$. Since both $\tau^{\text{system}}_{\text{IMU}}$ and $\tau^{\text{system}}_{\text{LiDAR}}$ are influenced by the non-deterministic computation duration of the OS, the selection of $\widehat{\mathbf{X}}$ then becomes non-deterministic, thus leading to non-reproducible DLIO. To ensure a deterministic fusion pair $(\mathbf{T}, \widehat{\mathbf{X}})$, the matrix $\mathbf{T}$ is stipulated to be fused with $\widehat{\mathbf{X}}$, whose timestamp τ_{IMU} is closest to the timestamp of $\mathbf{T}$ (i.e., τ_{LiDAR}). The timestamps τ_{IMU} and τ_{LiDAR} are deterministic and are related to the acquisition timestamp of sensor measurements.

B. A General Procedure

To identify previously unlisted non-deterministic implementations, we introduce a general procedure applicable across different LIO systems. Its effectiveness is validated by the successful identification of five such implementations. The procedure consists of the following steps:

1) Instrumentation of Key Code Blocks: Starting from a given LIO system, output statements (e.g., `std::cout`) are inserted immediately before the execution and immediately after the invocation of key functional code blocks (such as those implementing state propagation and state update) so as to log the variable (i.e., pose) values to a file.

2) Coarse Localization Via Repeated Executions: The modified system is then executed 30 times on given sequences, with the accuracy of each run recorded. If a difference in accuracy is detected between two consecutive runs, the executions are halted. The log files from these two runs are then compared to coarsely localize the code region responsible for numerical inconsistency and non-deterministic implementations. For example, if the values of variables remain consistent before the state update across two runs but differ afterward, non-deterministic implementations are likely located within the state update module.

3) Binary Search for Precise Localization: Suppose the coarsely identified code region spans from line L_{A} to line L_{B}. A binary search strategy is employed to narrow the range of lines containing non-deterministic implementations. Specifically, output statements are inserted at two positions: at the midpoint (i.e., L_{mid}) to record the values of variables in the subregion $[L_{\text{A}}, L_{\text{mid}}]$ and at L_B to record those in $[L_{\text{mid}}, L_{\text{B}}]$. These values are written to log files for analysis.

4) Execution and Comparison for Pinpointing: The modified system is then executed 30 times to evaluate localization accuracy. If a difference in accuracy is detected between any two consecutive runs, subsequent executions are terminated. The corresponding log files from the divergent runs are then compared. If differences in variable values are found within $[L_{\text{A}}, L_{\text{mid}}]$, the first line exhibiting the difference is identified as the non-deterministic line of code. Otherwise, the first line exhibiting the difference in $[L_{\text{mid}}, L_{\text{B}}]$ is identified as the non-deterministic line of code. Furthermore, if the identified line is a function call, the same binary search strategy can be recursively applied within the callee function to further pinpoint non-deterministic implementations.

5) Sequential Modification and Verification: Given the possibility that multiple non-deterministic implementations may contribute to non-reproducibility, each implementation is identified and modified sequentially. The earliest one, in chronological execution order, is first replaced with a deterministic version before proceeding to the subsequent non-deterministic ones. A system is considered reproducible once it produces consistent results with zero variance across 30 repeated runs on each sequence from the NCD [13], NCD++[14], and SubT-MRS [11] datasets.

TABLE I
SPECIFICATIONS OF LIO SYSTEMS AND DATASETS

Category	Name	Reference	LiDAR Configuration
LIO system	COIN-LIO [4]	*ICRA 2024*	–
	DLIO [5]	*ICRA 2023*	–
	FAST-LIO2 [6]	*TRO 2022*	–
	Faster-LIO [7]	RAL 2022	–
	iG-LIO [8]	*RAL 2024*	–
Dataset	NCD [13]	*ICRA 2022*	OS1-64
	NCD++ [14]	*arXiv 2022*	OS0-128
	SubT-MRS [11]	*CVPR 2024*	VLP-16
	ENWIDE [4]	*ICRA 2024*	OS0-128
	GEODE [15]	*arXiv 2024*	OS1-64, VLP-16

IV. EXPERIMENTAL SETUP

A. Systems and Datasets

The proposed five non-deterministic implementations can be found in the state-of-the-art LIO systems across five datasets. Brief specifications of these systems and datasets are summarized in Table I. Specifically:

1) **FAST-LIO2:** [6] employs point-to-plane tightly coupled optimization and incremental mapping.
2) **Faster-LIO:** [7] is an incremental improvement over FAST-LIO2, by introducing an alternative map data structure to enhance system efficiency.

TABLE II
HARDWARE AND SOFTWARE CONFIGURATIONS

Component	Specification
Processor	13th Gen Intel® Core™ i7-13700KF
OS Version	Ubuntu 20.04 LTS (64-bit)
Dependencies	The default versions of `ROS1`, `PCL`, `Eigen`, `Armadillo` are installed via `apt`.
Compiler Options	-O3 -std=c++17

3) **COIN-LIO:** [4] extends FAST-LIO2 by incorporating intensity information to improve system robustness against geometric degeneracy.
4) **DLIO:** [5] adopts loosely coupled optimization methods along with keyframe-based mapping.
5) **iG-LIO:** [8] applies GICP-based tightly coupled optimization and incremental mapping.

B. Runtime Environment

1) Hardware and Software Configuration: The hardware and software configurations are detailed in Table II.

2) Communication Protocols: To address the frame-drop-induced inconsistencies in the numbers of estimated poses across experimental runs, data streaming begins only after all subscribers of sensor data complete initialization. Additionally, each subscriber is set to maintain a 100,000-message queue to buffer LiDAR frames that cannot be processed at the incoming data rate (e.g., 10 Hz).

C. Algorithm Parameter Configurations

We do not use the default algorithm parameter configurations because we have observed that LIO systems with these configurations may consistently exhibit system breakdowns in challenging environments. These repeated failures do not provide meaningful results for analyzing system non-reproducibility. Therefore, we empirically fine-tune the parameters to the best of our ability.

D. Accuracy Evaluation

The non-reproducibility is evaluated in terms of accuracy, and the accuracy evaluation of poses estimated by LIO systems involves four key steps:

1) Pose interpolation: The ground truth (GT) poses whose timestamps t_{gt} fall within $[t_{\text{est}}^{\text{start}}, t_{\text{est}}^{\text{end}}]$ are interpolated at the timestamps of the estimated poses. Here, $t_{\text{est}}^{\text{start}}$ is the timestamp of the first estimated pose in an experiment, and $t_{\text{est}}^{\text{end}}$ is the timestamp of the final estimated pose. For interpolation, we apply linear interpolation to the translation components of the estimated poses and spherical linear interpolation to the rotation components.
2) (Coordinate) frame transformation: The estimated poses are transformed to align their child frame with the child frame of the GT poses using an extrinsic matrix, typically obtained through sensor calibration or obtained from a CAD model.
3) Pose alignment: The estimated and GT poses are associated through temporal nearest neighbor searches, with a maximum timestamp difference of 0.01 seconds. The poses in each associated pair are then aligned using a rigid transformation derived from the Umeyama [20] method.
4) Metric calculation: The root mean square error (RMSE) of the absolute translation errors (ATE) in meters, is used to evaluate the accuracy of LIO systems.

It should be noted that although modifying each LIO system to output poses at GT timestamps could theoretically yield higher accuracy in evaluation than pose interpolation, we intentionally adopt the interpolation method for practical considerations. Specifically, this choice preserves a plug-and-play evaluation paradigm while avoiding both intrusive and dataset-specific modifications.

TABLE III
ALIAS NAMES OF SYSTEM VARIANTS

System Variant	Non-Deterministic Implementation Described in				
	Sec.III-A1	Sec. III-A2	Sec. III-A3	Sec. III-A4	Sec. III-A5
COIN-LIO [4]	✓	✓	✓	–	–
COIN-LIO_100	✓	✗	✗	–	–
COIN-LIO_010	✗	✓	✗	–	–
COIN-LIO_001	✗	✗	✓	–	–
COIN-LIO_000	✗	✗	✗	–	–
DLIO [5]	–	–	–	✓	✓
DLIO_10	–	–	–	✓	✗
DLIO_01	–	–	–	✗	✓
DLIO_00	–	–	–	✗	✗
FAST-LIO2 [6]	✓	✓	–	–	–
FAST-LIO2_10	✓	✗	–	–	–
FAST-LIO2_01	✗	✓	–	–	–
FAST-LIO2_00	✗	✗	–	–	–
Faster-LIO [7]	–	✓	✓	–	–
Faster-LIO_10	–	✓	✗	–	–
Faster-LIO_01	–	✗	✓	–	–
Faster-LIO_00	–	✗	✗	–	–
iG-LIO [8]	–	–	–	✓	–
iG-LIO_0	–	–	–	✗	–

✓ indicates the presence of a non-deterministic implementation, while ✗ indicates that it has been replaced with a deterministic version.

V. EXPERIMENTAL RESULTS

In this section, each experiment was repeated 30 times using the setup described in Section IV. Additionally, each experiment involved running a LIO system on a sequence from a dataset and recording the system's accuracy. The system-dataset combinations used in these experiments were chosen from those listed in Table II. To distinguish original systems from their deterministic variants, alias names were assigned as shown in Table III.

A. Non-Deterministic Implementations as Dominant Causes of Non-Reproducibility

To demonstrate that non-deterministic implementations are the dominant causes of non-reproducibility under constant experimental conditions (detailed in Section II), we applied the solutions proposed in Section III-A to transform each evaluated system by replacing its non-deterministic implementations with deterministic counterparts. As expected, each of the resulting modified systems yielded consistent accuracy across repeated experiments on the NCD [13], NCD++[14], and Sub-MRS [11] datasets.

In addition, Figs. 5–7 present the accuracy variations of non-reproducibility caused by non-deterministic implementations. Specifically, low-magnitude non-reproducibility (i.e., variation

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.

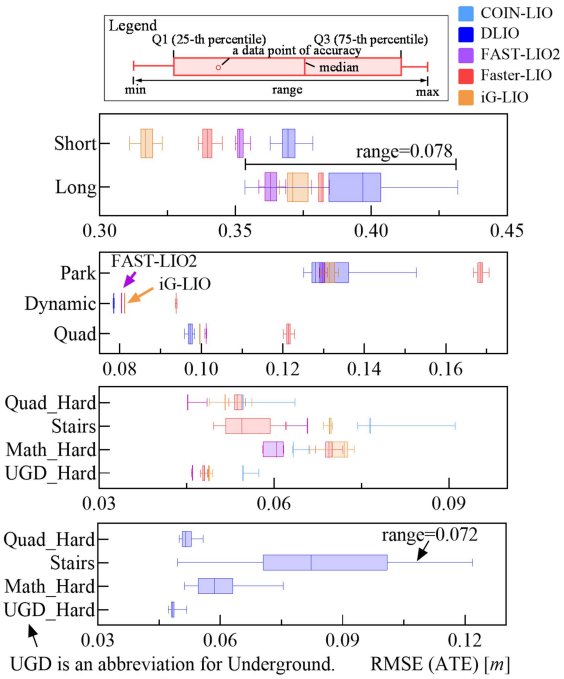


Fig. 5. Box plots illustrating the low-magnitude non-reproducibility of LIO systems. Although the boxes of different systems overlap, it is still evident that the range of each evaluated system remains below 0.10 m on both the NCD [13] and NCD++ [14] datasets.

TABLE IV
THE NON-REPRODUCIBILITY OF LIO SYSTEMS

System	Sequence	RMSE (ATE) [m] ↓ Min	Max	Range (Max-Min)
COIN-LIO [4]	RunwayS [4]	**1.022**	2.922	1.900
COIN-LIO_100	RunwayS	**1.022**	2.758	1.736
COIN-LIO_010	RunwayS	1.216	2.758	1.542
COIN-LIO_000	RunwayS	2.758	2.758	0.000
COIN-LIO	FieldD [4]	0.599	2.485	1.886
COIN-LIO_001	FieldD	**0.347**	1.863	1.516
COIN-LIO_000	FieldD	1.082	1.082	0.000
DLIO [5]	Sewerage [11]	**0.517**	3.087	2.570
DLIO_10	Sewerage [11]	0.527	0.527	0.000
DLIO_01	Sewerage [11]	0.521	7.012	6.490
DLIO_00	Sewerage [11]	0.527	0.527	0.000
FAST-LIO2 [6]	Sewerage [11]	**1.257**	4.962	3.705
FAST-LIO2_10	Sewerage [11]	**1.257**	2.443	1.186
FAST-LIO2_01	Sewerage [11]	1.617	4.962	3.345
FAST-LIO2_00	Sewerage [11]	1.617	1.617	0.000
Faster-LIO [7]	Sewerage	0.649	4.667	4.018
Faster-LIO_10	Sewerage	0.646	2.492	1.846
Faster-LIO_01	Sewerage	**0.616**	4.936	4.320
Faster-LIO_00	Sewerage	2.492	2.492	0.000

The same information is presented in a graphical form in Fig. 7.

below 10 cm) is more common in datasets collected with high-channel mechanical LiDARs, such as the 64-channel LiDAR used in the NCD dataset and Stairs_β sequence, and the 128-channel LiDAR in NCD++ (Figs. 5–6). However, even with high-channel LiDARs, geometrically degenerate environments, such as the flat fields in the FieldD sequence, can make noteworthy non-reproducibility observable (Fig. 7). Furthermore, Figs. 6–7 suggest that noteworthy non-reproducibility tends to occur more frequently in datasets captured with low-channel LiDARs (e.g., the 16-channel LiDAR used in SubT-MRS and Stairs_α) than with high-channel LiDARs.

In conclusion, while non-deterministic implementations can affect non-reproducibility, the effect becomes more evident in datasets collected by low-channel (i.e., low-vertical-resolution) LiDARs or in geometrically degenerate environments.

A possible explanation for the observed low-magnitude accuracy is that high-channel LiDARs offer vertically dense LiDAR scans in non-degenerate environments. These scans may help accurate pose estimation by providing reliable local geometric features such as plane normals (used in [4], [6], [7]) and plane covariances (used in [5], [8]). As a result, LIO systems become less sensitive to non-deterministic implementations, even when such implementations involve software defects (see Section II-I-A3).

B. Ablation Studies

We further conducted ablation studies on LIO systems with multiple non-deterministic implementations to evaluate the individual impact of each implementation on system non-reproducibility. Table IV and Fig. 7 show that for each of the five identified types of non-deterministic implementation, there exists a LIO system where one of these implementations independently causes noteworthy non-reproducibility. To explain this phenomenon, we offer several preliminary insights: (1) Small changes in nearest neighbor selection (see Section III-A1) may lead to erroneous geometric correspondences used for pose optimization, as it is often assumed that nearby points or planes belong to the same object (an assumption that may not hold), thus causing errors to propagate and accumulate. (2) Although changes in floating-point addition order (see Section III-A4) are expected to cause minor numerical differences, smaller than 1×10^{-9}, between the results obtained from different addition orders in iG-LIO and DLIO, some systems (e.g., iG-LIO) exhibit unexpectedly large accuracy variations beyond 1 m, unlike systems such as DLIO. Regarding iG-LIO, to the best of our knowledge, we have not found any race conditions in its implementation of multi-threaded reduction. (3) Thread-unsafe implementations (see Section III-A3) and non-deterministic state fusion (see Section III-A5) are essentially software defects which must be corrected. (4) Table IV and Fig. 7 also show that a non-deterministic implementation can occasionally lead to higher accuracy than the deterministic counterpart.

C. Mechanism Analysis

Recall that Section III-A2 suggests that hardware-dependent implementations can result in inconsistent numerical outputs and non-reproducibility. However, the underlying causal mechanism remains unexplored. This section validates our hypothesis that the hardware-dependent results observed in `Eigen` -based matrix multiplication (see Section III-A2) may arise from differences in the addition order during multiplication. The hypothesis is supported if deliberately modifying the addition order, while keeping the execution core fixed, leads to observable non-reproducibility.

To test this hypothesis, for the systems COIN-LIO_000, FAST-LIO2_000, and Faster-LIO_000, we alter the addition order of elements in $\mathbf{H}^\top\mathbf{H}$ by left-multiplying $\mathbf{H}$ with a randomly

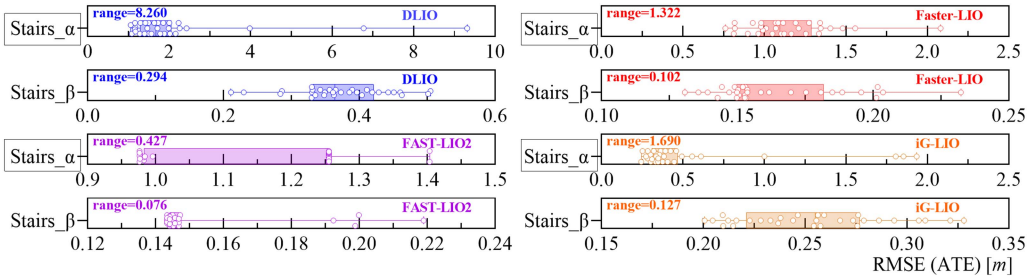


Fig. 6. Box plots illustrating the non-reproducibility of LIO systems in the GEODE [15] dataset. In each subplot, the boxes, lines, and circle points follow the standard convention (see Fig. 5) to represent the distributions of RMSEs for the non-reproducible system (labeled in the top-right corner). The Stairs_α and Stairs_β sequences were collected in the same stairway environment but with different LiDAR configurations: a 16-channel mechanical LiDAR for Stairs_α and a 64-channel mechanical LiDAR for Stairs_β. The one-order-of-magnitude RMSE difference of a system between the Stairs_α and Stairs_β sequences indicates that system non-reproducibility becomes more pronounced in datasets collected using low-channel LiDARs.

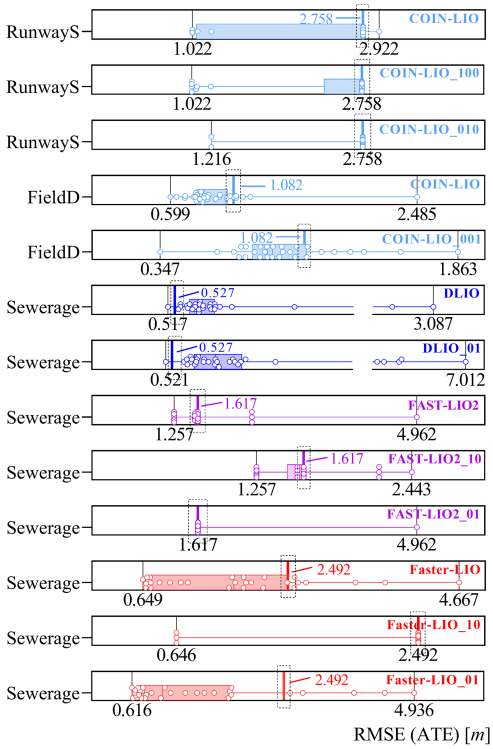


Fig. 7. Box plots illustrating the non-reproducibility of LIO systems (the same information is presented in a tabular form in Table IV). In each subplot, the boxes, lines, and circle points follow the standard convention (see Fig. 5) to represent the distributions of RMSEs for the non-reproducible system (labeled in the top-right corner), while the **bold** line within the dashed box indicates the RMSE of the reproducible system. The visualization illustrates that one of the non-deterministic implementations independently induces noteworthy non-reproducibility and that non-reproducible systems can sometimes achieve higher accuracy than reproducible ones.

generated permutation matrix $\mathbf{P}$, resulting in a row-permuted matrix $\mathbf{H}_{\text{perm}} = \mathbf{PH}$ (see Listing 2). Note that the row permutation only changes the addition order in $\mathbf{H}^\top\mathbf{H}$ and does not change the result of the matrix multiplication under exact arithmetic, ensuring that $\mathbf{H}^\top\mathbf{H} = \mathbf{H}_{\text{perm}}^\top\mathbf{H}_{\text{perm}}$. In practical numerical computations of our experiments, we observe that the Frobenius norm of $\mathbf{H}^\top\mathbf{H} - \mathbf{H}_{\text{perm}}^\top\mathbf{H}_{\text{perm}}$ remains consistently below 1×10^{-9}.

The results in Fig. 8 validate the hypothesis. Specifically, the systems COIN-LIO_000, FAST-LIO2_000, and Faster-LIO_000, which initially exhibited RMSE above 1 m, achieved

```
MatrixXd calculateHTH(int seed, const MatrixXd& H) {
+ MatrixXd H_tmp = H;
+ PermutationMatrix<Dynamic, Dynamic> permutation(H.rows());
+ permutation.setIdentity();
+ std::mt19937 g(seed);
+ std::shuffle(permutation.indices().data(),
+ permutation.indices().data() + permutation.size(), g);
+ H_tmp = permutation * H_tmp;
+ return H_tmp.transpose() * H_tmp;
- return H.transpose() * H;
}
```

Listing 2. Simplified code modifications marked (+ additions / − deletions) for shuffling addition order in computation $\mathbf{H}^\top\mathbf{H}$, relative to the original implementation [6]. Note that the code snippet contains only the algorithm's critical logic, with auxiliary code omitted for clarity.

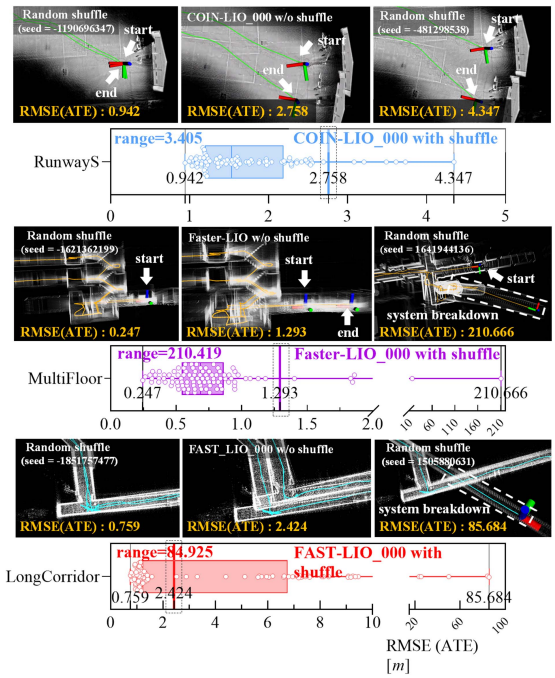


Fig. 8. The effect of the addition order in computing $\mathbf{H}^\top\mathbf{H}$ (mentioned in Section III-A2) on non-reproducibility. The experiment was repeated 100 times to ensure sufficient coverage of abnormal cases. Each accuracy measurement in the box plot corresponds to a distinct addition order determined by a random row permutation. The visualization illustrates that a reproducible system with shuffling may exhibit either higher accuracy (see the images in the first column) or lower accuracy (see the images in the third column) compared to a reproducible one without shuffling (see the images in the second column).

sub-meter accuracy after applying the shuffle operation. In certain instances, however, the same operation resulted in system breakdowns. Furthermore, in the MultiFloor sequence [11], assuming NV-LIOM [21] is evaluated under the same methodology as used in our experiments, it is noteworthy that FAST-LIO2_000, when using an addition order shuffled with the seed -1621362199, achieved an RMSE of 0.247, outperforming the reported 0.254 RMSE of NV-LIOM. This observation is particularly impressive considering that NV-LIOM is specifically designed for indoor environments and incorporates a loop closure module.

VI. Conclusion

LIO systems seem to exhibit noteworthy non-reproducibility in challenging datasets where LiDAR scans feature low vertical resolution (e.g., 2°) or are collected in geometrically degenerate scenarios (e.g., flat fields). Such non-reproducibility could be attributed to non-deterministic implementations, such as non-deterministic nearest neighbors and non-deterministic matrix multiplication. Remarkably, simply shuffling the addition order in matrix multiplication has been demonstrated to produce accuracy variations beyond $1\,\mathrm{m}$ as well.

While the presented experimental results indicate that non-deterministic implementations cause non-reproducibility under constant experimental conditions, a rigorous mathematical explanation of this causal relationship has yet to be established and is left for future work. Additionally, although the proposed five non-deterministic implementations in this work do not encompass all possible non-deterministic ones in current LIO systems, their revelation remains important. Specifically, this work (1) lays the groundwork for identifying the sources of non-reproducibility in LIO systems and (2) alerts researchers to the potential risks of drawing unreliable conclusions when performing comparative evaluations or incremental studies on systems with such implementations. We hope our experimental findings caution LIO researchers against drawing false-positive conclusions in the presence of the noteworthy non-reproducibility.

References

[1] D. Merkel, "Docker: Lightweight Linux containers for consistent development and deployment," *Linux J.*, vol. 239, Mar. 2014, Art. no. 2.

[2] Y. Yang, B. Xu, Y. Li, and S. Schwertfeger, "The SLAM hive benchmarking suite," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2023, pp. 11257–11263.

[3] N. Radulov, Y. Zhang, M. Bujanca, R. Ye, and M. Luján, "A framework for reproducible benchmarking and performance diagnosis of SLAM systems," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2024, pp. 14225–14232.

[4] P. Pfreundschuh, H. Oleynikova, C. Cadena, R. Siegwart, and O. Andersson, "COIN-LIO: Complementary intensity-augmented LiDAR inertial odometry," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2024, pp. 1730–1737.

[5] K. Chen, R. Nemiroff, and B. T. Lopez, "Direct LiDAR-inertial odometry: Lightweight LIO with continuous-time motion correction," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2023, pp. 3983–3989.

[6] W. Xu, Y. Cai, D. He, J. Lin, and F. Zhang, "FAST-LIO2: Fast direct LiDAR-inertial odometry," *IEEE Trans. Robot.*, vol. 38, no. 4, pp. 2053–2073, Aug. 2022.

[7] C. Bai, T. Xiao, Y. Chen, H. Wang, F. Zhang, and X. Gao, "Faster-LIO: Lightweight tightly coupled LiDAR-inertial odometry using parallel sparse incremental voxels," *IEEE Robot. Automat. Lett.*, vol. 7, no. 2, pp. 4861–4868, Apr. 2022.

[8] Z. Chen, Y. Xu, S. Yuan, and L. Xie, "iG-LIO: An incremental GICP-based tightly-coupled LiDAR-inertial odometry," *IEEE Robot. Automat. Lett.*, vol. 9, no. 2, pp. 1883–1890, Feb. 2024.

[9] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, Jan. 2014.

[10] E. D. Cubuk, B. Zoph, D. Mané, V. Vasudevan, and Q. V. Le, "AutoAugment: Learning augmentation strategies from data," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 113–123.

[11] S. Zhao et al., "SubT-MRS dataset: Pushing SLAM towards all-weather environments," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2024, pp. 22647–22657.

[12] X. Bouthillier, C. Laurent, and P. Vincent, "Unreproducible research is reproducible," in *Proc. 36th Int. Conf. Mach. Learn.*, K. Chaudhuri and R. Salakhutdinov, Eds., Jun. 2019, pp. 725–734.

[13] M. Ramezani, Y. Wang, M. Camurri, D. Wisth, M. Mattamala, and M. Fallon, "The newer college dataset: Handheld LiDAR, inertial and vision with ground truth," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2020, pp. 4353–4360.

[14] L. Zhang, M. Camurri, D. Wisth, and M. Fallon, "Multi-camera LiDAR inertial extension to the newer college dataset," 2021, *arXiv:2112.08854*.

[15] Z. Chen et al., "Heterogeneous LiDAR dataset for benchmarking robust localization in diverse degenerate scenarios," *Int. J. Robot. Res.*, 2025.

[16] K. Huang, J. Zhao, Z. Zhu, C. Ye, and T. Feng, "LOG-LIO: A LiDAR-inertial odometry with efficient local geometric information estimation," *IEEE Robot. Automat. Lett.*, vol. 9, no. 1, pp. 459–466, Jan. 2024.

[17] *IEEE Standard for Floating-Point Arithmetic*, IEEE Standard 754-2019, 2019, pp. 1–84.

[18] D. Goldberg, "What every computer scientist should know about floating-point arithmetic," *ACM Comput. Surv.*, vol. 23, no. 1, pp. 5–48, Mar. 1991.

[19] K. Chen, B. T. Lopez, A.-A. Agha-mohammadi, and A. Mehta, "Direct LiDAR odometry: Fast localization with dense point clouds," *IEEE Robot. Automat. Lett.*, vol. 7, no. 2, pp. 2000–2007, Apr. 2022.

[20] S. Umeyama, "Least-squares estimation of transformation parameters between two point patterns," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 13, no. 4, pp. 376–380, Apr. 1991.

[21] D. Chung and J. Kim, "NV-LIOM: LiDAR-inertial odometry and mapping using normal vectors towards robust SLAM in multifloor environments," *IEEE Robot. Automat. Lett.*, vol. 9, no. 11, pp. 9375–9382, Nov. 2024.