Determinism Analysis in Hybrid-LLM-GNN Modeling for Materials Property Prediction

Youjia Li[1], Daniel Wines[2], Kamal Choudhary[2,3,4], Vishu Gupta[1,5,6], Muhammed Nur Talha Kilic[7], Sayak Chakrabarty[7], Wei-keng Liao[1], Alok Choudhary[1], Ankit Agrawal[1*]

[1] Department of Electrical and Computer Engineering, Northwestern University, Evanston, IL, USA

[2] Material Measurement Laboratory, National Institute of Standards and Technology, 100 Bureau Dr, Gaithersburg, MD, USA

[3] Department of Materials Science and Engineering, Whiting School of Engineering, The Johns Hopkins University, Baltimore, MD 21218, USA

[4] Department of Electrical and Computer Engineering, Whiting School of Engineering, The Johns Hopkins University, Baltimore, MD 21218, USA

[5] Lewis-Sigler Institute for Integrative Genomics, Princeton University, Princeton, NJ, USA

[6] Ludwig Institute for Cancer Research, Princeton University, Princeton, NJ, USA

[7] Department of Computer Science, Northwestern University, Evanston, IL, USA

*Author to whom any correspondence should be addressed

E-mail: ankit-agrawal@northwestern.edu

Abstract.

Driven by advances in artificial intelligence and the growing availability of databases, machine learning now plays a central role in data-driven materials knowledge discovery. In studies that employ machine learning models, maintaining deterministic workflows is crucial for reproducibility, as it ensures that repeated runs with identical settings yield consistent and reliable predictions. Here, we conduct a determinism analysis of the recently proposed hybrid LLM-GNN framework for material property prediction. Our study provides a comprehensive analysis of sources of non-determinism, with particular focus on variability introduced by hardware variations and software stacks across training and inference pipelines. By distinguishing different forms of determinism, we assess how discrepancies arise under varying execution conditions. In cases where results are non-deterministic, we further analyze and compare prediction differences to assess the impact of these variations. Our investigation uncovers systematic patterns in prediction discrepancies at the dataset level as well as variations at the individual sample level. Based on the findings, we identify the main sources of divergence and provide practical recommendations to improve deterministic behavior in ML-based materials property prediction workflows.

1. Introduction

Advances in artificial intelligence and the expansion of databases have transformed how researchers uncover scientific knowledge, as machine learning (ML) has become an indispensable tool in data-driven scientific discovery [1, 2, 3, 4]. In scientific AI and ML workflows, reproducibility generally refers to the ability of an independent research team to obtain the same results using the original authors' released artifacts under the same experimental setup [5, 6], such that the same computational procedures can be followed, corroborating results can be produced, and similar scientific conclusions can be drawn [7]. For material property prediction, reproducible ML model outputs support the reliable description of structure-property relationships, which further guide subsequent computational exploration and experimental screening. Reproducibility can be influenced by multiple factors, including the accessibility of datasets and source code, as well as the transparency, documentation, and completeness of the released machine learning pipeline [8, 9, 10, 11]. Among these factors, deterministic behavior is a fundamental mechanism for ML model reproducibility, as it ensures that identical inputs, model parameters, and configurations, when executed in the same environment, consistently produce bit-exact outputs across repeated runs. Beyond reproducibility, determinism also facilitates reliable model comparisons and supports iterative development across varying modeling choices, including input representations, feature constructions, and architectural designs.

In ML models, violations of determinism lead to run-to-run variation. Non-determinism can arise from several sources. Some sources are intrinsic, such as data shuffling, weight initialization, and dropout, and can be mitigated through seed control to achieve single-environment determinism. However, the same does not hold for other extrinsic system-level factors that limit cross-environment determinism. Floating-point non-associativity causes numerical discrepancies in matrix calculations under parallel execution [12, 13]. This property means that the result of arithmetic operations may depend on the order of evaluation, even if mathematically equivalent (e.g., $(a+b)+c \neq a+(b+c)$). In addition, deterministic implementations of deep learning operations remain incomplete, as frameworks like PyTorch [14] do not provide deterministic variants for all algorithms. These behaviors often result from backend libraries or hardware-specific kernel optimizations, which are generally beyond the user's control.

Despite the practical significance and technical challenges, there has been a lack of deterministic analysis in ML for material science, particularly as state-of-the-art models grow more complex and diverse to enhance model predictive performance. To the scientific community, reproducibility has received attention with most efforts focusing on validating result consistency across independent reruns and reproductions [15, 16, 17]. In particular, Li et al. [18] conduct a reproducibility analysis of the ALIGNN graph neural network by re-evaluating model performance under repeated training runs with different random seeds. In addition to assessments, multiple studies have offered broader recommendations for combating the reproducibility crisis, including those by Stodden

et al. [19], Isdahl et al. [20], Henderson et al. [21], Heil et al. [22], and several others. Following these reproducibility-focused efforts, Heumos et al. [23] dig deeper by explicitly analyzing deterministic behavior in bioinformatics ML pipelines. Their study and the follow-up work [24] evaluate hardware variation with the focus on an end-to-end manner and do not separate training and inference stages or quantify the level of numerical discrepancies relative to other sources of variation. Our study is complementary to these efforts, as it provides a focused analysis of deterministic behavior within scientific ML workflows for materials property prediction, explicitly separating training- and inference-stage effects, comparing discrepancies induced by hardware and deep learning frameworks, and relating dataset-level consistency to individual sample-level shifts that may influence downstream screening.

Recently, both Graph Neural Networks (GNNs) and Large Language Models (LLMs) have been increasingly adopted in materials science discovery. For the task of crystal property prediction, GNNs excel at capturing local structural features from atomic graphs, while LLMs offer strong generalization by encoding global knowledge from a large corpus of scientific text. There have been several studies [25, 26, 27, 28, 29, 30, 31] that explore the potential of combining the strength of GNN-based models and LLM models to integrate textual information from natural language with structure-aware learning from GNNs for computational materials modeling. Despite the modeling advantages of such fusion approaches, the added architectural complexity introduces new sources of computational variability and obscures deterministic behavior. Our study is motivated by the need to systematically analyze sources of non-determinism in ML-based data mining workflows used in materials science, and to provide practical guidance for obtaining deterministic results. Here we present a case study aiming to analyze the deterministic behavior of the hybrid LLM-GNN framework [25] by Li *et al.* to enhance crystal property prediction. As one of the first multimodal learning frameworks that integrates structural and textual modalities for materials property prediction, the aforementioned work combines graph-based embeddings from the Atomistic Line Graph Neural Network (ALIGNN) model [32] with contextual language embeddings from MatBERT [33], a domain-specific adaptation of the Bidirectional Encoder Representations from Transformers (BERT) model pre-trained on materials science literature. ALIGNN captures local atomic environments and geometric dependencies within crystal structures, while MatBERT extracts semantic representations from text descriptions generated for the same materials. These complementary embeddings are concatenated to form a unified feature vector, which is then passed to a predictive neural network model.

The remainder of this paper is organized as follows. Section 2 presents the overall workflow and methodology for enforcing determinism and details the setup of controlled experiments for deterministic analysis. Section 3 presents the experimental results and analyzes patterns of non-determinism arising from software frameworks, GPU parallelism, and cross-environment hardware differences. Finally, Section 4 summarizes the observed sources and patterns of non-deterministic behavior and concludes with

practical guidelines for achieving determinism.

2. Methodology

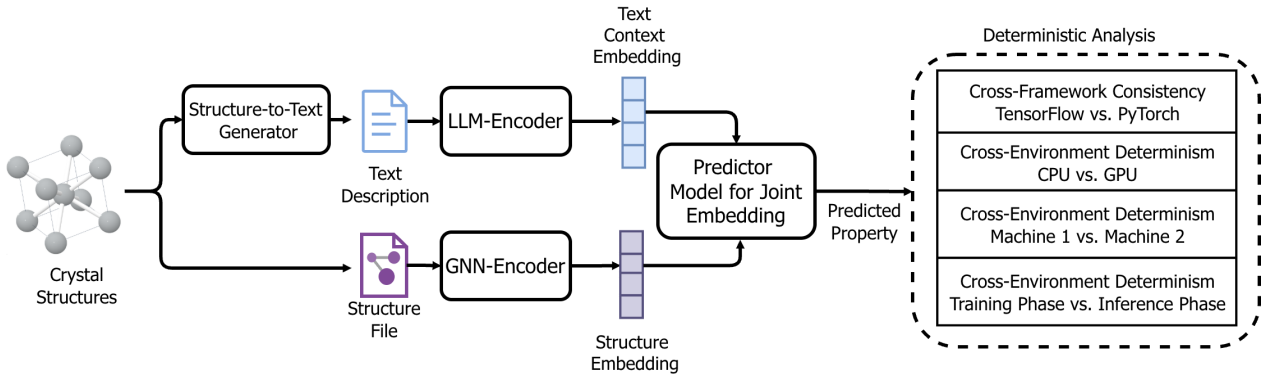

Figure 1. Workflow of Hybrid-LLM-GNN transfer learning for determinism analysis. First, the input structure file is fed to a text generator to produce domain knowledge descriptions. Next, the pre-trained LLM, serving as a knowledge model, is employed to extract contextual word embeddings. The downstream hybrid-modal predictor model integrates the embeddings and serves as the final predictor for material property estimation. We probe the final prediction outcomes and analyze deterministic behavior from four perspectives.

2.1. Hybrid-LLM-GNN Framework

The hybrid modeling workflow used for determinism evaluation is adopted from [25]. As illustrated in Figure 1, it combines both structural and textual representations for materials property prediction. Graph-based embeddings are generated by using the ALIGNN model trained on formation energy data from the Materials Project (MP). In a separate thread, the same input structure file is fed to a text generator to produce domain knowledge descriptions. These string representations are piped into a pre-trained language model. The original study investigates multiple combinations of text generators and pre-trained models. In this work, we focus solely on evaluating determinism based on Robocrystallographer [34] and MatBERT [33]. The ALIGNN and MatBERT embeddings are concatenated and fed into a fully connected neural network for downstream prediction. The final prediction outcomes are examined to assess deterministic behavior.

2.2. Setup for Determinism Assessment

Before presenting the experimental setup for deterministic analysis, we clarify the three levels of numerical consistency behavior examined in this work:

- **Single-environment determinism** refers to the ability of the same code to produce bit-exact results across repeated runs on the same machine under an

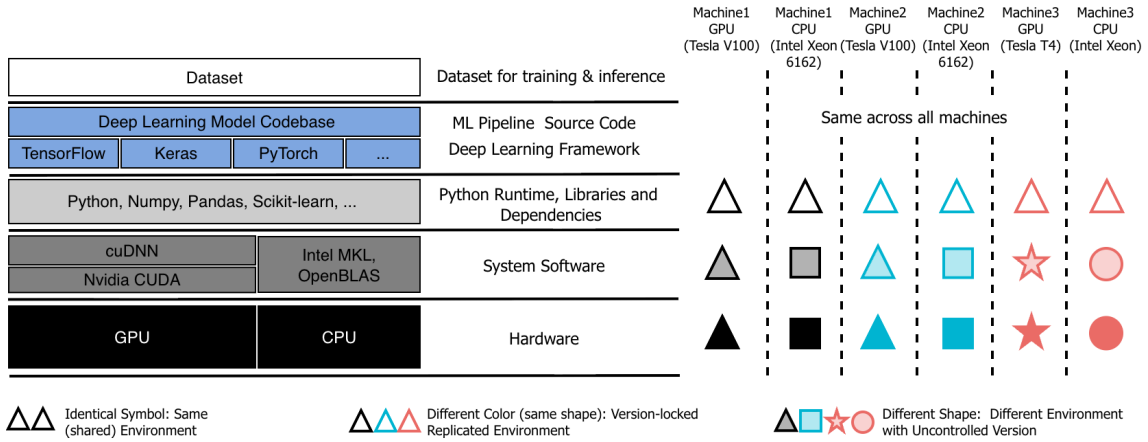

Figure 2. Machine learning software and hardware stacks with an illustration of differences in the environments we set up for controlled experiments. Machine 1 runs both GPU and CPU executions in the same shared Python environment. Machine 2 replicates Machine 1 with a version-locked reinstalled Python environment and identical hardware (CPU and GPU) model. Machine 3 uses a version-locked Python environment but differs in the hardware model, introducing a controlled hardware variation. CUDA (Compute Unified Device Architecture), cuDNN (CUDA Deep Neural Network library), MKL (Intel Math Kernel Library), and OpenBLAS (Open Basic Linear Algebra Subprograms) are backend libraries responsible for low-level numerical and tensor operations.

identical software and hardware stack. This form of determinism can generally be achieved through appropriate deterministic configurations and represents the most commonly discussed determinism, with explicit support provided by modern deep learning frameworks.

- **Cross-environment determinism** refers to the ability of the same code to produce bit-exact results across different environments, especially under hardware variations. It extends single-environment determinism beyond a single physical machine by examining consistency across environments. Modern deep learning frameworks provide no explicit guarantee for this form of determinism.
- **Cross-framework consistency** refers to the agreement between models implemented in different deep learning frameworks, such as PyTorch and TensorFlow, using the same deep learning model architecture, identical datasets, and matched hyperparameters. In this setting, discrepancies from framework-specific implementations and backend libraries are unavoidable, and consistency is assessed in terms of the magnitude of numerical differences rather than strict bit-exact equality.

To assess the deterministic behavior of the aforementioned Hybrid-LLM-GNN framework, we systematically explored configurable options to eliminate sources of stochastic behavior. Notably, certain components of the pipeline are inherently deterministic. The texts generated by Robocrystallographer are the exact same

sentences over repeated runs on the same input. The embedding extractor uses pretrained GNN and LLM models in inference mode without training or fine-tuning. We set the models to evaluation mode to deactivate stochastic layers such as dropout. With this setup, the generated embeddings remain static in the same hardware environment.

The focus of examining different configurations to study determinism in this work is on the downstream predictor, which is trained in an end-to-end manner in the above pipeline. We first set the random seed used in the plain Python random generator, NumPy random functions, the deep learning framework (e.g., PyTorch), and GPU CUDA operations. This ensures deterministic behavior in model initialization, data shuffling, and GPU-accelerated computations over repeated runs. Secondly, we enabled deterministic behavior in the backend by disabling algorithm benchmarking and enforcing the use of deterministic computation paths, which helps eliminate non-determinism from GPU kernel selection and low-level operations. With these configurations, we verified single-environment determinism by executing multiple repeated runs and confirmed bit-exact results at practical precision. Unless otherwise specified, all experiments reported in this study satisfy single-environment determinism, and repeated runs therefore yield identical results and are not separately reported. On the basis of these configurations, we applied additional constraints to meet the requirements of the controlled experiments. For deep learning framework-level comparisons, all results were computed on the same machine with GPU acceleration (Machine 1 GPU). For the CPU vs. GPU and cross-environment deterministic behavior experiments, we locked the entire software stack by building all models in PyTorch and fixing the versions of all other Python libraries to ensure consistency across software environments.

To evaluate cross-environment determinism, we deployed the workflow on three machines representing different tiers of environmental control. Here we use the term "environment" to refer to a stack that can include software and hardware components, unlike the typical use, where it often refers only to software. Figure 2 illustrates the software and hardware stacks involved for a typical machine learning workflow, along with the environmental differences across three machines. At the top level, model development relies on a codebase and associated datasets, implemented using high-level deep learning frameworks such as TensorFlow, Keras, and PyTorch. These frameworks operate within a Python runtime environment that includes foundational libraries such as NumPy, Pandas, and Scikit-learn. Most well-documented machine learning codebases provide sufficient information to enable replication and reproduction up to this level. Beneath this layer, system-level support libraries such as cuDNN and CUDA for GPU acceleration and Intel MKL or OpenBLAS for CPU computation will handle numerical execution with optimization. These lower layers typically fall outside the control of machine learning researchers during reproduction. Among the environments set for our controlled experiments, Machine 1 GPU serves as the baseline environment. Machine 1 CPU performs executions in the same physical Python environment, but branches off starting from system-level libraries. Machine 2 is a hardware-identical twin of

Machine 1, equipped with the same CPU and GPU models. However, Machine 2 has to employ a freshly reinstalled Python environment reconstructed using version-locked specifications. Machine 3 also adopts a version-locked replicated Python environment, but it differs from Machines 1 and 2 in the system-level libraries and GPU hardware model. On the right side, we use symbolism to show the nuanced differences at each stack across machines. In this figure, different shapes denote distinct environments with uncontrolled versions, while different colors of the same shape denote a version-locked replicated environment. Identical symbols indicate the same physical environments, such as cases where both CPU and GPU training on the same machine share the exact same installed Python, rather than separate installations with the same version. By "version-locked Python environment," we refer to reconstructing the software stack using an identical dependency specification file (e.g., requirements.txt) and reinstalling all packages via pip install, with pinned Python version and library versions (e.g., PyTorch 2.0.0), which is the standard practice adopted in most ML model codebases. With this experimental design, we investigated how deterministic behavior can be affected by hardware discrepancies.

3. Results and Discussions

3.1. Dataset

The materials datasets used in this study are generated from first-principles calculations based on Density Functional Theory (DFT). We use the same DFT-computed materials datasets as in the original model for the deterministic analysis. The downstream property prediction tasks are performed using multiple target properties from the 2022.12.12 release of JARVIS-3D [35, 36, 37], which contains 75,993 materials. Target properties to predict include formation energy per atom (Formation Energy), energy above the hull (Ehull), magnetic moment (Magout), modified Becke Johnson potential (MBJ) band gaps [38] ($\text{Bandgap}_{\text{mBJ}}$), topological spin-orbit spillage (Spillage) [39], spectroscopic limited maximum efficiency (SLME) [40] and superconducting transition temperature (Tc_supercon) [41].

3.2. Numerical Consistency under Deep Learning Framework Variation

Our first investigation focuses on cross-framework numerical consistency. Both PyTorch and TensorFlow can be configured to yield bit-exact deterministic results at the selected display precision across repeated runs within each framework. However, when comparing outputs across frameworks, their predictions diverge due to differences in the underlying implementations of the two frameworks.

This cross-framework investigation into numerical consistency shows that the level of divergence varies across target properties, as summarized in Table 1. The Mean Absolute Error (MAE) values from PyTorch and TensorFlow remain at a similar scale but are not bit-exact, and the Mean Absolute Difference (MAD) of the prediction

difference between the two frameworks tends to be proportional to the MAE level, ranging from 39% to 51% of its value, which suggests that the level of prediction discrepancy is correlated with the model's accuracy performance. This aligns with our expectation that more challenging prediction tasks lead to greater model uncertainty when evaluated on the test set. Tasks that are more difficult for the model to learn, such as predicting Tc_supercon, tend to exhibit higher modeling uncertainty, which increases sensitivity to numerical variation caused by different deep learning frameworks. As a result, less accurate models show greater discrepancies across frameworks.

Table 1. Comparison of MAE (between prediction and ground truth) values and Prediction MAD (between PyTorch and TensorFlow prediction) for ALIGNN-MatBERT-based transfer learning using PyTorch and TensorFlow implementations across various material properties. Tested Material properties include: formation energy per atom (Formation Energy), energy above the hull (Ehull), magnetic moment (Magout), modified Becke Johnson potential (MBJ) band gaps ($\text{Bandgap}_{\text{mBJ}}$), topological spin-orbit spillage (Spillage), spectroscopic limited maximum efficiency (SLME), and superconducting transition temperature (Tc_supercon).

Property	Data Size	PyTorch MAE	TensorFlow MAE	Prediction MAD
Formation Energy (eV atom^{-1})	75799	0.0350	0.0339	0.0176
Ehull (eV atom^{-1})	74926	0.0360	0.0357	0.0167
Magout (μ_{B})	74212	0.4116	0.4211	0.2096
$\text{Bandgap}_{\text{MBJ}}$ (eV)	19556	0.2416	0.2516	0.1164
Spillage	11301	0.3135	0.3140	0.1245
SLME (%)	9762	4.4510	4.5634	2.1494
Tc_supercon (K)	1054	2.0378	2.1095	0.8850

To further assess the consistency between frameworks beyond aggregate error metrics, we examine the prediction distributions directly. Figure 3 presents the parity plots for four representative properties, while the remaining properties are shown in Figure S1 in the Supplementary Information (SI). As shown in Figure 3, the predictions for formation energy are not bit-exact, but both the scatter plot and the marginal histograms indicate that the outputs from the two frameworks are highly consistent and closely aligned. However, for properties with poor model accuracy such as T_{c}, their prediction distributions show a larger spread in the parity plots, with more points deviating from the reference line.

Bland-Altman plots are typically employed to compare the difference between two measurements for identifying consistent offsets or trends in the differences across the range of measured values. Here, we apply this method to evaluate the agreement between predictions from PyTorch and TensorFlow models across multiple target properties. The Bland-Altman plots in Figure 4 show that the mean of the prediction differences between PyTorch and TensorFlow is not always close to zero across all target properties. Some properties exhibit noticeable shifts. For example, Spillage has a mean prediction difference of 0.124, indicating that PyTorch tends to produce higher predictions than

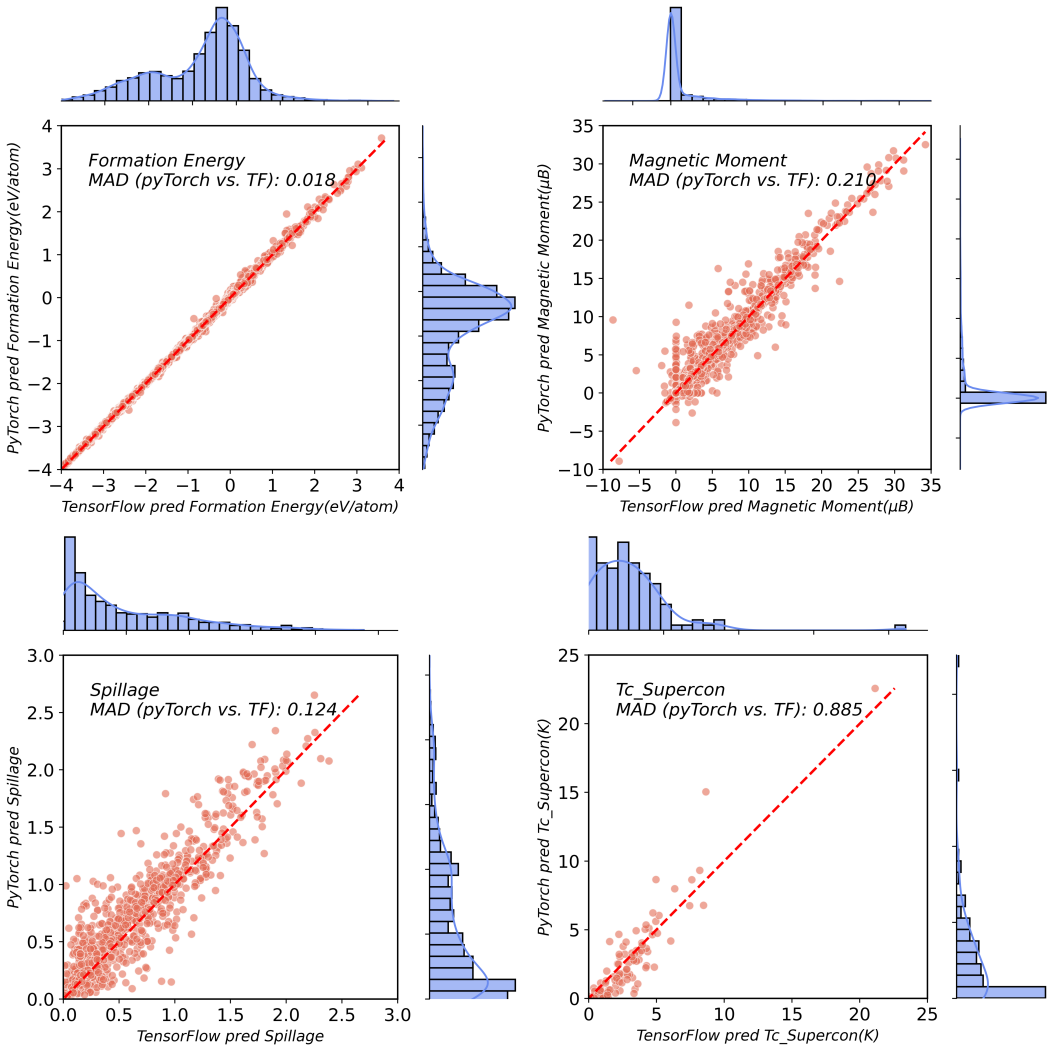

Figure 3. Parity plot for model predictions from ALIGNN-MatBERT-based transfer learning using PyTorch and TensorFlow implementations in the same computing environment. The marginal histogram plot displays the distribution of predictions. Tested Material properties include: formation energy per atom (Formation Energy), magnetic moment (Magout), topological spin-orbit spillage (Spillage), and superconducting transition temperature (Tc_supercon).

TensorFlow for this property. In addition, we observe a linear boundary in the distribution of prediction differences for Magnetic moment and Spillage, which indicates the presence of proportional bias, where the magnitude of the prediction difference between frameworks increases with the property value.

3.3. Deterministic Behavior under Hardware Variation

As a first step toward analyzing cross-environment determinism, we examine the impact of GPU parallelism on deterministic behavior by comparing end-to-end executions on the CPU and GPU within the same machine when all software stacks remain identical. We

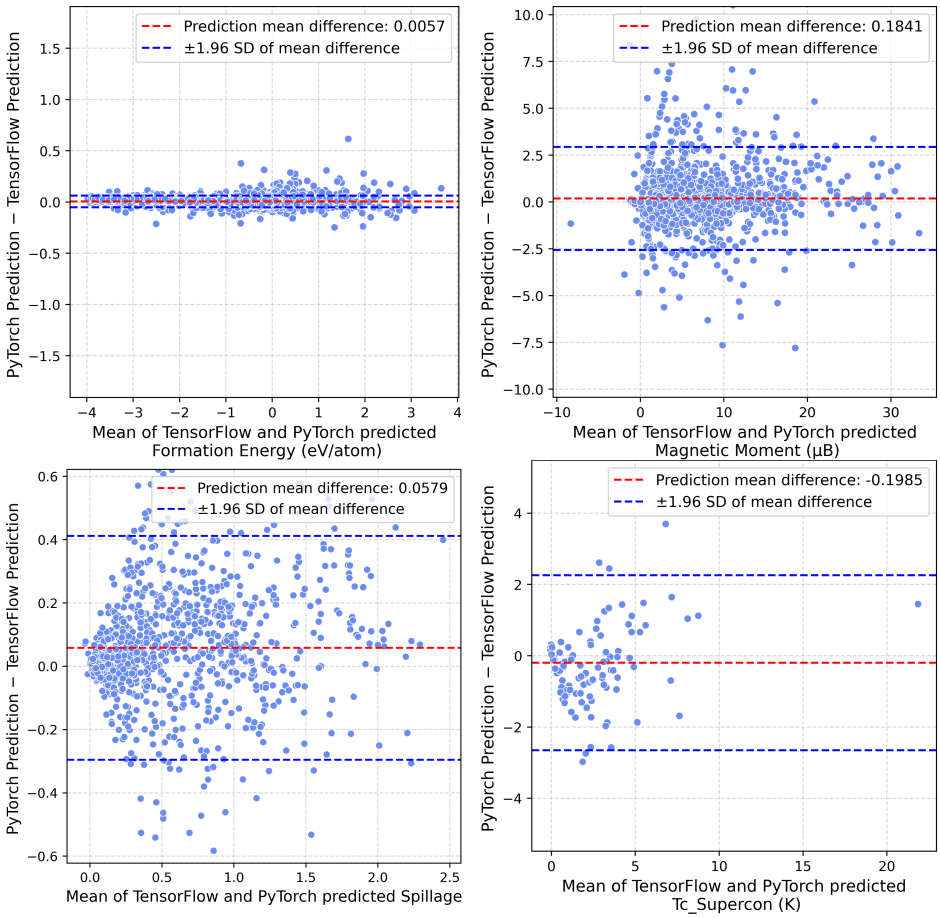

Figure 4. Bland-Altman plot for model predictions from ALIGNN-MatBERT-based transfer learning using PyTorch and TensorFlow implementations in the same computing environment. Tested Material properties include: formation energy per atom (Formation Energy), magnetic moment (Magout), topological spin-orbit spillage (Spillage), and superconducting transition temperature (Tc_supercon).

Table 2. Comparison of MAE (between prediction and ground truth) values for ALIGNN-MatBERT-based transfer learning across various computing environments. Tested Material properties include: modified Becke Johnson potential (MBJ) band gaps and superconducting transition temperature (Tc_supercon). Results that are bit-exact within the selected precision are marked with the same superscript symbol.

Training & Inference Environment	Hardware Model	Tc_supercon (K)	$\text{Bandgap}_{\text{MBJ}}$ (eV)
Machine 1 GPU	Tesla V100-PCIE	2.0378444195^{*}	$0.2415528595^{\S}$
Machine 1 CPU	Intel Xeon 6126	$2.0379059315^{\dagger}$	$0.245349288^{\ddagger}$
Machine 2 GPU	Tesla V100-PCIE	2.0378444195^{*}	$0.2415528595^{\S}$
Machine 2 CPU	Intel Xeon 6126	$2.0379059315^{\dagger}$	$0.245349288^{\ddagger}$
Machine 3 GPU	Tesla T4	2.0286376476	0.2471545786
Machine 3 CPU	Intel Xeon	2.0266072750	0.247285977

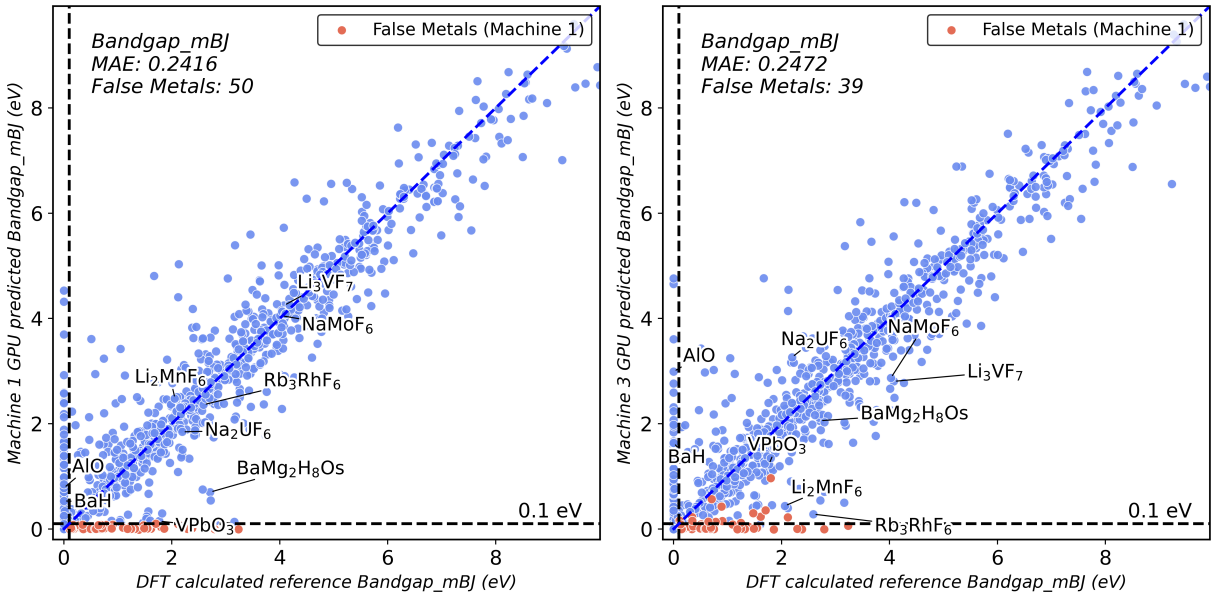

Figure 5. Parity plots comparing model predictions for band gaps under different experimental conditions (left: Machine 1 GPU; right: Machine 3 GPU). Predicted band gaps are plotted against DFT-calculated reference band gaps. The Mean Absolute Error (MAE) and the number of false metals (predicted band gap < 0.1 eV) are reported in each panel. False metal samples predicted on Machine 1 are highlighted in red in both plots.

conducted end-to-end training and testing of the same model on two machines, one using CPU and the other using GPU acceleration, to compare prediction differences across compute hardware. With the configuration described in Section 2.2, both CPU and GPU executions produce deterministic results across repeated runs on their respective backends. However, the CPU and GPU outputs differ from each other, even on the same machine. As shown in Table 2, on all three machines, the outputs from the CPU and GPU pairs are not bit-exact when compared to each other. This discrepancy arises primarily from non-determinism introduced by parallel kernel execution on GPUs. Specifically, the order of operations in parallel reductions can vary from the sequential order in the CPU, which leads to small but measurable numerical differences. As a result, the GPU-accelerated version often produces a distinct output variant compared to the CPU version.

We then evaluated the consistency of model outputs by performing end-to-end training and testing on different machines to assess how hardware differences affect determinism. As summarized in Table 2, the MAE values from Machine 1 and Machine 3 are not bit-exact. Deterministic results are not guaranteed when executing the same model on different systems, even with identical software stacks, due to subtle variations in hardware behavior. However, when two systems share the same CPU and GPU hardware models, bit-exact outputs can be achieved at the selected display precision, as demonstrated by the bit-exact results between Machine 1 and Machine 2. Machine 1 and

Machine 2 are equipped with the same CPU and GPU hardware models, which result in bit-exact MAEs between their CPU executions and between their GPU executions. This finding is confirmed on another hybrid LLM-GNN model, MatMMFuse [29], using the same dataset and repeated experimental protocol. Due to the large model size and long training time, we conducted these additional experiments only on the GPUs of Machine 1, Machine 2, and Machine 3. Table S1 in the SI summarizes the MAE results across the three hardware environments. As shown in Table S1, the same GPU hardware models generate bit-exact MAE values at the reported precision.

In cases where cross-environment determinism does not hold, it is also valuable to reference these execution-environment divergences in terms of their practical scientific impact in downstream contexts. To further examine the practical scientific impact of execution-environment divergences, we analyze the false metal count under two hardware environments: Machine 1 GPU and Machine 3 GPU. The concept of false metals is commonly used in band gap benchmarks to quantify cases where non-metal materials are predicted with near-zero band gaps. In this study, we adopt a threshold of 0.1 eV to tag such cases for analysis, recognizing that this value represents a pragmatic cutoff for near-zero predictions rather than a definitive classification of metallicity. The parity plots in Figure 5 summarize both the false metal counts and the distributional differences under these two conditions. When changing the execution environment from Machine 1 to Machine 3, the false metal count decreases from 50 to 39, while the MAE shifts slightly from 0.2416 to 0.2472. Although these dataset-level metric shifts appear modest in magnitude, the impact at the individual sample level can be substantial. Comparing the highlighted samples across the two plots shows that a noticeable portion of samples tagged as false metals under Machine 1 predictions move above the threshold in the Machine 3 predictions and are therefore no longer tagged as false metals. At the same time, several samples exhibit substantial shifts in predicted band gap across hardware environments. For example, the prediction for Rb_3RhF_6 changes from 2.36 eV to 0.28 eV. These sample-level shifts indicate that hardware-dependent numerical variation can be large enough to exceed tolerance and flip the decision for an individual sample in downstream screening.

To better understand the causes of output inconsistency across environments observed above, we separately examine the effects of training and inference when performed on different hardware. Table 3 summarizes the test set MAE for ALIGNN-MatBERT-based transfer learning predictions across six computing environments, with each model trained and then inferred on potentially different machines. Noticeably, the variation is largely driven by the training environment rather than the inference environment. Within each column, the MAE values are either identical to the self-inference case within the selected precision or differ only beyond the eighth decimal place. In contrast, when comparing MAE values within the same row, the differences are more pronounced, with discrepancies appearing as early as the 3rd decimal place. This finding aligns with the known sources of cross-machine non-determinism. Numerical differences accumulate with the number of floating-point operations, and since inference

Table 3. Test set MAE (between prediction and ground truth) for ALIGNN-MatBERT-based transfer learning across six different computing environments, evaluated on the modified Becke Johnson (MBJ) band gap property. Each model is trained in one environment and then evaluated (inference) in another. Diagonal entries represent cases (in bold) where training and inference were performed on the same machine as in Table 2.

	Training Environment					
Inference Environment	Machine 1 GPU	Machine 1 CPU	Machine 2 GPU	Machine 2 CPU	Machine 3 GPU	Machine 3 CPU
Machine 1 GPU	**0.2415528595**	0.2453492582	0.2415528595	0.2453492582	0.2471545786	0.2472859770
Machine 1 CPU	0.2415528595	**0.2453492880**	0.2415528595	0.2453492880	0.2471545786	0.2472859770
Machine 2 GPU	0.2415528595	0.2453492582	**0.2415528595**	0.2453492582	0.2471545786	0.2472859770
Machine 2 CPU	0.2415528595	0.2453492880	0.2415528595	**0.2453492880**	0.2471545786	0.2472859770
Machine 3 GPU	0.2415528595	0.2453492582	0.2415528595	0.2453492582	**0.2471545786**	0.2472859919
Machine 3 CPU	0.2415528595	0.2453492880	0.2415528595	0.2453492880	0.2471545786	**0.2472859770**

involves only the forward pass without backpropagation, it is less affected by hardware-induced variability. This pattern is further confirmed by inspecting the model weights from all six training environments, which show numerical differences in weights between any two models trained on different machines (e.g., Machine 1 CPU vs Machine 1 GPU), as shown in Figure S3 and Figure S4 in the SI. The same trend is observed for the MatMMFuse model (see Table S2 in the SI), further confirming that variations in the training environment have a dominant influence on deterministic behavior. We also examined the raw predictions for a single sample using the same setup shown in Table 4. For the MgB_2 sample, predictions across 36 training-inference combinations exhibit a consistent pattern: the training environment has a dominant influence on the predicted value, while hardware differences during inference have negligible impacts. Interestingly, the numerical identity in the reported MAE does not always correspond to bit-exact agreement in the raw predictions for individual samples. For example, in Table 3, the MAEs for the model trained on Machine 1 GPU (leftmost column) are bit-exact at the selected display precision across all inference environments. However, the corresponding single-sample predictions are not bit-exact and show slight numerical variation across all values in this column. This inconsistency arises because the non-determinism at the individual sample level is extremely small and partially cancels out when aggregated to the dataset-level MAE, which leads to seemingly identical values within the displayed decimal precision for MAE.

3.4. Summary

We summarize the sources of numerical discrepancies by comparing the relative impact of controlled variations in deep learning frameworks, training and inference machines, use of GPU acceleration, and inference environments on model predictions. In this analysis, the predictions from the PyTorch model trained and inferred on the GPU

Table 4. Predicted modified Becke Johnson potential (MBJ) band gap across various computing environments for the MgB_2 sample. Each prediction is based on a model trained in one environment and then evaluated (inference) on another among all six environments. Diagonal entries (in bold) represent cases where training and inference were performed on the same machine.

	Training Environment					
Inference Environment	Machine 1 GPU	Machine 1 CPU	Machine 2 GPU	Machine 2 CPU	Machine 3 GPU	Machine 3 CPU
Machine 1 GPU	**-0.000584588**	0.000259697	-0.000584588	0.000259697	-0.003095329	-0.000541091
Machine 1 CPU	-0.000584647	**0.000259638**	-0.000584647	0.000259638	-0.003095239	-0.000541106
Machine 2 GPU	-0.000584588	0.000259697	**-0.000584588**	0.000259697	-0.003095329	-0.000541091
Machine 2 CPU	-0.000584647	0.000259638	-0.000584647	**0.000259638**	-0.003095239	-0.000541106
Machine 3 GPU	-0.000584602	0.000259668	-0.000584602	0.000259668	**-0.003095284**	-0.000541121
Machine 3 CPU	-0.000584647	0.000259638	-0.000584647	0.000259638	-0.003095239	**-0.000541106**

of Machine 1 are used as the reference. Predictions are then compared against four alternative setups that isolate specific sources of variation: using TensorFlow on the same machine, running on a different GPU machine, executing on the CPU only, and performing inference on the CPU after GPU training. As shown in the MAD values in Figure 6, the level of deviation from the reference predictions gradually decreases from the first condition to the fourth condition. The model trained using a different framework (TensorFlow vs PyTorch) introduces the largest prediction deviation, with a prediction MAD of 0.116. Hardware differences between CPU and GPU, whether on the same machine or across machines, lead to similar levels of variation, with slightly lower MAD values compared to the framework level difference. In contrast, using a different hardware at the inference stage alone has a negligible effect. The prediction MAD is extremely small, and the scatter plot shows data points tightly aligned along the reference line. We also summarize the results for the T_c prediction task, as shown in Figure S5 in the SI, which exhibits the same overall trend.

Despite the above numerical differences introduced at the ML modeling stage, these variations should be examined from a broader scientific perspective by situating them within the methodological uncertainty associated with the underlying DFT reference calculations. When examined through dataset-level aggregated statistics, the magnitude of ML-induced numerical discrepancies is overall less dominant than the intrinsic uncertainty present at the DFT stage. One primary source of uncertainty arises from methodological differences. For band gap, prior benchmark studies [42, 43, 44] show that discrepancies across different exchange-correlation approximations and computational schemes within DFT can lead to variations ranging from 0.4-1.7 eV in MAE when compared against the reference experimental band gaps. The second source of uncertainty stems from convergence-related parameters within DFT calculations, such as k-point density, q-point sampling, and electronic broadening. For the superconducting transition temperature, varying the electronic broadening parameter alone can change

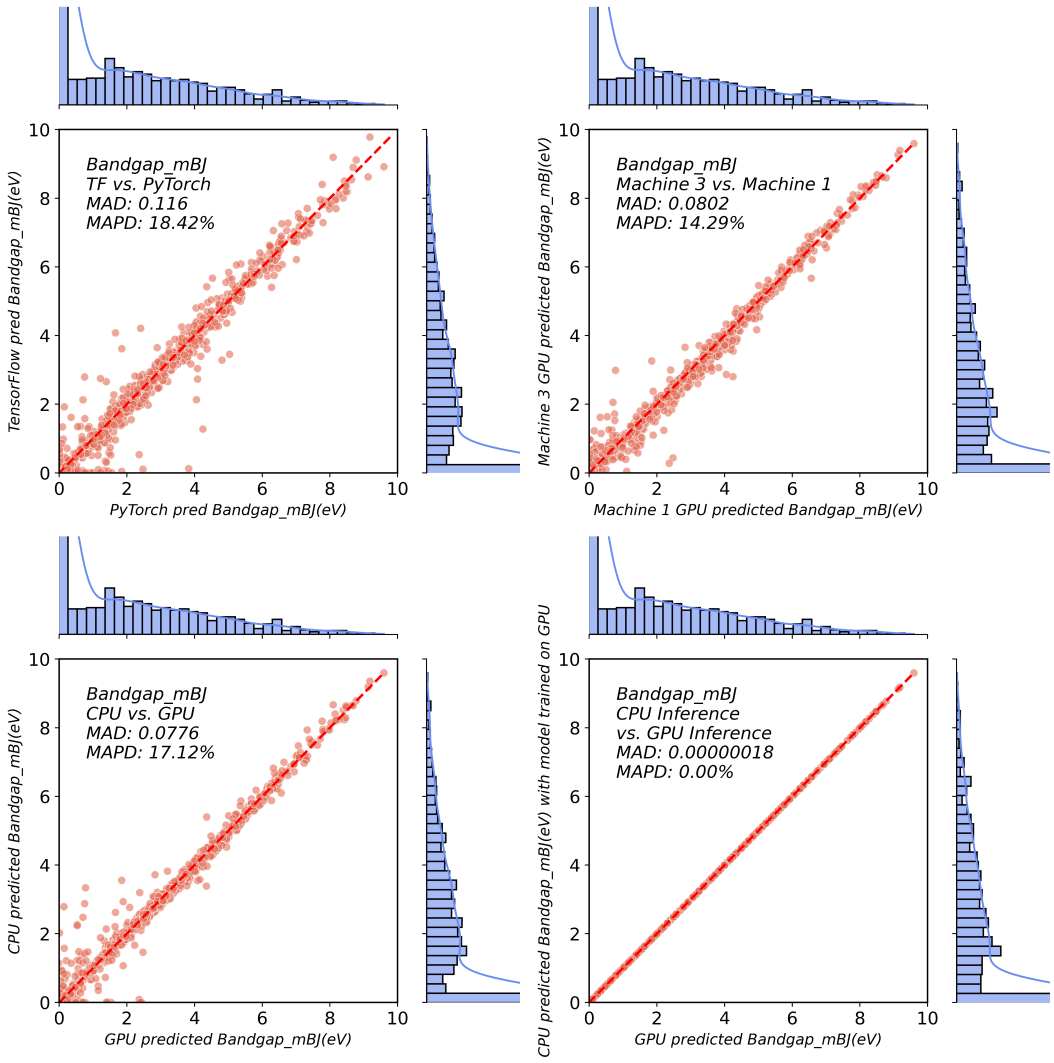

Figure 6. Parity plots comparing model predictions for the modified Becke Johnson potential (MBJ) band gap from ALIGNN-MatBERT-based transfer learning under different experimental conditions. The x-axis in all subplots shows predictions from a PyTorch model trained and tested on the GPU of Machine 1. From top left to lower right (in row-major order), the y-axis introduces different sources of variation: (1) TensorFlow implementation on the same machine (2) predictions with model inference on a different GPU machine, (3) predictions from a model on the same machine but trained and tested on the CPU, and (4) predictions from the same trained model but with inference on the CPU instead of the GPU. The Mean Absolute Difference (MAD) and the Mean Absolute Percentage Difference (MAPD) of predictions are reported in each panel. MAPD is computed using samples with band gap values > 0.1 eV to avoid near-zero reference values that can lead to inflated percentage errors.

the MAE of computed T_c by nearly 10K relative to experiment [41]. Kawamura et al. [45] showed that with carefully tuned DFT configurations, the computed T_c values can still deviate by up to $\pm 20\%$ from experimental reference data. In comparison, the sub-kelvin numerical differences in MAE for T_c caused by hardware variation in Table 2 are negligible. More broadly, there are other sources of instability in

computational materials workflows, such as molecular dynamics (MD) simulations, where finite simulation time, mean squared displacement (MSD) fitting window, and thermostat settings can introduce variability in computed properties [46, 47, 48]. For example, a previous molecular dynamics study reports that the error in the estimated diffusion coefficient can be about 9-17%, depending on the trajectory length and block size used for statistical averaging [46].

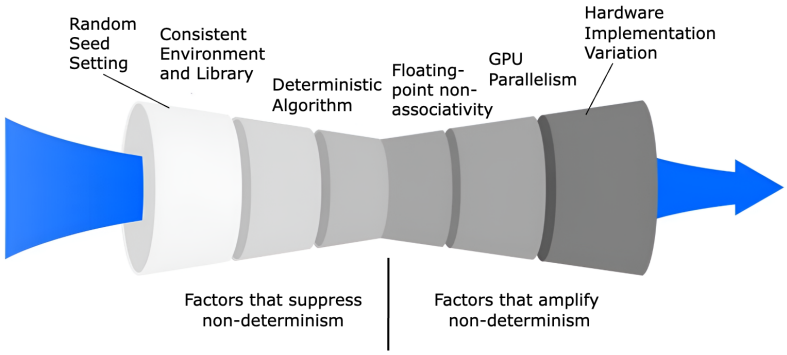

Figure 7. Funnel diagram illustrating factors affecting determinism in deep learning workflows. The left half of the figure represents factors that help suppress non-determinism, including seed setting, consistent environments, and deterministic algorithms. The right half shows the progressive accumulation of non-deterministic effects caused by floating-point non-associativity, GPU parallelism, and hardware implementation variation.

4. Conclusion

This study provides a systematic analysis of numerical consistency in ML-based materials property prediction workflows. Our findings show that single-environment determinism is straightforward to achieve through fixed random seeds and the use of deterministic operations. In contrast, cross-environment determinism is more challenging, as bit-exact consistency across different computing environments is not guaranteed unless the CPU or GPU hardware models are identical across machines. Among hardware-induced variations, the training environment plays a dominant role, while models trained on the same machine typically produce near bit-exact predictions across different inference environments. In addition, cross-framework consistency between implementations in different deep learning frameworks, such as PyTorch and TensorFlow, exhibits larger numerical discrepancies than hardware variation under the same codebase, reflecting inherent differences in backend implementations and numerical kernels. The numerical differences introduced by ML frameworks or computing environments can be small in magnitude at the dataset level, especially when compared with the intrinsic uncertainty of DFT calculations. Nonetheless, the prediction shifts

at the individual sample level can still be substantial and may affect one sample's downstream materials screening.

Based on our findings, determinism in modern ML pipelines in scientific discovery like material property prediction can be influenced by a combination of suppressing and amplifying factors throughout the training and inference pipeline. We summarized practices that help enforce determinism and identified sources of nondeterminism that introduce variability and cause divergence in the results. As depicted in Figure 7, deterministic behavior on the same machine can be achieved by setting fixed random seeds, using a consistent Python and software environment, and selecting deterministic algorithm implementations when supported. Despite these operations, nondeterministic behavior may still occur due to the following factors that cannot be fully controlled in cross-environment reproduction. One of the primary causes for this is floating-point non-associativity [12, 13], which causes numerical differences depending on the order of arithmetic operations. GPU acceleration introduces additional variation due to non-deterministic kernel execution in parallel reductions. Hardware model differences across systems also lead to inconsistencies, even when the software stack remains the same. These factors together limit the extent to which deep learning workflows can produce deterministic results across arbitrary machines. Based on our findings, we provide a set of recommendations for enforcing determinism and reproducible results:

Set random seeds for all stochastic operations, including data shuffling, weight initialization, and dropout layers.

Disable non-deterministic algorithms by enabling deterministic flags in the deep learning framework when available (e.g., `torch.use_deterministic_algorithms` in PyTorch), and avoid using layers or components that lack deterministic implementations.

Lock software versions including the Python runtime, deep learning framework, associated dependencies, and system-level support libraries such as CUDA, as much as possible to minimize variation caused by software stacks.

Explicitly log hardware configurations such as GPU hardware models, to document potential sources of numerical differences and support reproducibility.

Expect numerical differences when running end-to-end training and testing across different machines, as most non-determinism is introduced during the training or fine-tuning process. Model inference on pre-trained models is expected to show negligible cross-environment variation.

A natural direction for future work is to extend this analysis to other scientific ML workflows that accelerate materials discovery beyond regression tasks, including high-throughput screening surrogates and inverse design pipelines based on generative AI. In particular, property-conditioned generative models and closed-loop optimization frameworks may exhibit different levels of sensitivity to small numerical perturbations, as minor shifts in prediction scores can influence candidate ranking and selection. Such

investigations would provide a more comprehensive understanding of how numerical discrepancies propagate through end-to-end ML-based materials discovery pipelines.

5. Data Availability

The datasets used in this paper are publicly available from the corresponding websites-MP4 from https://materialsproject.org/, JARVIS from https://jarvis.nist.gov.

6. Code Availability

The codebase used in this study to perform deterministic analysis is available at `https://github.com/Jonathanlyj/Deterministic_Hybrid-LLM-GNN`.

7. Acknowledgments

All authors thank the National Institute of Standards and Technology for funding, computational, and data-management resources. Certain commercial equipment, instruments, software, or materials are identified in this paper in order to specify the experimental procedure adequately. Such identifications are not intended to imply recommendation or endorsement by NIST, nor is it intended to imply that the materials or equipment identified are necessarily the best available for the purpose. This work was performed under the following financial assistance award 70NANB24H136 from U.S. Department of Commerce, National Institute of Standards and Technology as part of the Center for Hierarchical Materials Design (CHiMaD). Partial support is also acknowledged from NSF award OAC-2331329 and Northwestern Center for Nanocombinatorics.

Conflicts of interest

There are no conflicts to declare.

Ethics Statement

This work does not involve human participants, personally identifiable data, or experiments with animals. All experiments were conducted on publicly available datasets and open-source software frameworks in accordance with the publication's ethical standards.

References

[1] Ankit Agrawal and Alok Choudhary. Perspective: Materials informatics and big data: Realization of the "fourth paradigm" of science in materials science. *Apl Materials*, 4:5, 2016.

[2] Tony Hey, Stewart Tansley, Kristin Michele Tolle, et al. *The fourth paradigm: data-intensive scientific discovery*, volume 1. Microsoft research Redmond, WA, 2009.

[3] Pervasive machine learning in physics. *Nature Reviews Physics*, 4:353, 2022.

[4] Vishu Gupta, Wei-keng Liao, Alok Choudhary, and Ankit Agrawal. Evolution of artificial intelligence for application in contemporary materials science. *MRS communications*, 13(5):754–763, 2023.

[5] Hans E Plesser. Reproducibility vs. replicability: a brief history of a confused terminology. *Frontiers in neuroinformatics*, 11:76, 2018.

[6] Association for Computing Machinery. Artifact review and badging. `https://www.acm.org/publications/policies/artifact-review-and-badging-current`, 2023. Accessed: 2026-02-05.

[7] Steven N Goodman, Daniele Fanelli, and John PA Ioannidis. What does research reproducibility mean? *Science translational medicine*, 8(341):341ps12–341ps12, 2016.

[8] Matthew Hutson. Artificial intelligence faces reproducibility crisis, 2018.

[9] Anthony Yu-Tung Wang, Ryan J Murdock, Steven K Kauwe, Anton O Oliynyk, Aleksander Gurlo, Jakoah Brgoch, Kristin A Persson, and Taylor D Sparks. Machine learning for materials scientists: an introductory guide toward best practices. *Chemistry of Materials*, 32(12):4954–4965, 2020.

[10] Nongnuch Artrith, Keith T Butler, François-Xavier Coudert, Seungwu Han, Olexandr Isayev, Anubhav Jain, and Aron Walsh. Best practices in machine learning for chemistry. *Nature chemistry*, 13(6):505–508, 2021.

[11] Iqbal Misbah, Carman KM Lee, and Kin Lok Keung. Fault diagnosis in rotating machines based on transfer learning: Literature review. *Knowledge-Based Systems*, 283:111158, 2024.

[12] Willow Ahrens, James Demmel, and Hong Diep Nguyen. Algorithms for efficient reproducible floating point summation. *ACM Transactions on Mathematical Software (TOMS)*, 46(3):1–49, 2020.

[13] Sanjif Shanmugavelu, Mathieu Taillefumier, Christopher Culver, Oscar Hernandez, Mark Coletti, and Ada Sedova. Impacts of floating-point non-associativity on reproducibility for hpc and deep learning applications (2024). *URL https://arxiv. org/abs/2408.05148*, 2024.

[14] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[15] Christian Collberg and Todd A Proebsting. Repeatability in computer systems research. *Communications of the ACM*, 59(3):62–69, 2016.

[16] Odd Erik Gundersen and Sigbjørn Kjensmo. State of the art: Reproducibility in artificial intelligence. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[17] Vishu Gupta, Wei-Keng Liao, Alok Choudhary, and Ankit Agrawal. Which deep learning framework should i use: A comparative study for deep regression modeling. In *2022 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 72–77. IEEE, 2022.

[18] Kangming Li, Brian DeCost, Kamal Choudhary, and Jason Hattrick-Simpers. A reproducibility study of atomistic line graph neural networks for materials property prediction. *Digital Discovery*, 3(6):1123–1129, 2024.

[19] Victoria Stodden, Marcia McNutt, David H Bailey, Ewa Deelman, Yolanda Gil, Brooks Hanson, Michael A Heroux, John PA Ioannidis, and Michela Taufer. Enhancing reproducibility for computational methods. *Science*, 354(6317):1240–1241, 2016.

[20] Richard Isdahl and Odd Erik Gundersen. Out-of-the-box reproducibility: A survey of machine learning platforms. In *2019 15th international conference on eScience (eScience)*, pages 86–95. IEEE, 2019.

[21] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Proceedings of the AAAI conference on artificial*

intelligence, volume 32, 2018.

[22] Benjamin J Heil, Michael M Hoffman, Florian Markowetz, Su-In Lee, Casey S Greene, and Stephanie C Hicks. Reproducibility standards for machine learning in the life sciences. *Nature methods*, 18(10):1132–1135, 2021.

[23] Lukas Heumos, Philipp Ehmele, Luis Kuhn Cuellar, Kevin Menden, Edmund Miller, Steffen Lemke, Gisela Gabernet, and Sven Nahnsen. mlf-core: a framework for deterministic machine learning. *Bioinformatics*, 39(4):btad164, 2023.

[24] Julian Wanner, Luis Kuhn Cuellar, Luiselotte Rausch, Kenneth W Berendzen, Friederike Wanke, Gisela Gabernet, Klaus Harter, and Sven Nahnsen. Nf-root: A best-practice pipeline for deep-learning-based analysis of apoplastic ph in microscopy images of developmental zones in plant root tissue. *Quantitative Plant Biology*, 5:e12, 2024.

[25] Youjia Li, Vishu Gupta, Muhammed Nur Talha Kilic, Kamal Choudhary, Daniel Wines, Wei-keng Liao, Alok Choudhary, and Ankit Agrawal. Hybrid-llm-gnn: integrating large language models and graph neural networks for enhanced materials property prediction. *Digital Discovery*, 2025.

[26] Janghoon Ock, Joseph Montoya, Daniel Schweigert, Linda Hung, Santosh K Suram, and Weike Ye Unimat. Unifying materials embeddings through multi-modal learning. *arXiv preprint arXiv:2411.08664*, 2024.

[27] Jaewan Lee, Changyoung Park, Hongjun Yang, Sungbin Lim, and Sehui Han. Cast: Cross attention based multimodal fusion of structure and text for materials property prediction. *arXiv preprint arXiv:2502.06836*, 2025.

[28] Kishalay Das, Pawan Goyal, Seung-Cheol Lee, Satadeep Bhattacharjee, and Niloy Ganguly. Crysmmnet: multimodal representation for crystal property prediction. In *Uncertainty in Artificial Intelligence*, pages 507–517. PMLR, 2023.

[29] Abhiroop Bhattacharya and Sylvain G Cloutier. Matmmfuse: Multi-modal fusion model for material property prediction. *arXiv preprint arXiv:2505.04634*, 2025.

[30] Janghoon Ock, Srivathsan Badrinarayanan, Rishikesh Magar, Akshay Antony, and Amir Barati Farimani. Multimodal language and graph learning of adsorption configuration in catalysis. *Nature Machine Intelligence*, 6(12):1501–1511, 2024.

[31] Jiao Huang, Qianli Xing, Jinglong Ji, and Bo Yang. Code-generated graph representations using multiple llm agents for material properties prediction. In *Forty-second International Conference on Machine Learning*.

[32] Kamal Choudhary and Brian DeCost. Atomistic line graph neural network for improved materials property predictions. *npj Computational Materials*, 7(1):185, 2021.

[33] Nicholas Walker, Amalie Trewartha, Haoyan Huo, Sanghoon Lee, Kevin Cruse, John Dagdelen, Alexander Dunn, Kristin Persson, Gerbrand Ceder, and Anubhav Jain. The impact of domain-specific pre-training on named entity recognition tasks in materials science. *Available at SSRN 3950755*, 2021.

[34] Alex M Ganose and Anubhav Jain. Robocrystallographer: automated crystal structure text descriptions and analysis. *MRS Communications*, 9(3):874–881, 2019.

[35] Kamal Choudhary, Kevin F Garrity, Andrew CE Reid, Brian DeCost, Adam J Biacchi, Angela R Hight Walker, Zachary Trautt, Jason Hattrick-Simpers, A Gilad Kusne, Andrea Centrone, et al. The joint automated repository for various integrated simulations (jarvis) for data-driven materials design. *npj computational materials*, 6(1):173, 2020.

[36] Daniel Wines, Ramya Gurunathan, Kevin F Garrity, Brian DeCost, Adam J Biacchi, Francesca Tavazza, and Kamal Choudhary. Recent progress in the jarvis infrastructure for next-generation data-driven materials design. *Applied Physics Reviews*, 10(4), 2023.

[37] Kamal Choudhary. The jarvis infrastructure is all you need for materials design. *Computational Materials Science*, 259:114063, 2025.

[38] Kamal Choudhary, Qin Zhang, Andrew CE Reid, Sugata Chowdhury, Nhan Van Nguyen, Zachary Trautt, Marcus W Newrock, Faical Yannick Congo, and Francesca Tavazza. Computational screening of high-performance optoelectronic materials using optb88vdw and tb-mbj formalisms.

Scientific data, 5(1):1–12, 2018.

[39] Kamal Choudhary, Kevin F Garrity, and Francesca Tavazza. High-throughput discovery of topologically non-trivial materials using spin-orbit spillage. *Scientific reports*, 9(1):8534, 2019.

[40] Kamal Choudhary, Marnik Bercx, Jie Jiang, Ruth Pachter, Dirk Lamoen, and Francesca Tavazza. Accelerated discovery of efficient solar cell materials using quantum and machine-learning methods. *Chemistry of materials*, 31(15):5900–5908, 2019.

[41] Kamal Choudhary and Kevin Garrity. Designing high-tc superconductors with bcs-inspired screening, density functional theory, and deep-learning. *npj Computational Materials*, 8(1):244, 2022.

[42] Max Großmann, Marc Thieme, Malte Grunert, and Erich Runge. Many-body perturbation theory vs. density functional theory: a systematic benchmark for band gaps of solids. *npj Computational Materials*, 2026.

[43] Fabien Tran, Jan Doumont, Leila Kalantari, Peter Blaha, Tomáš Rauch, Pedro Borlido, Silvana Botti, Miguel AL Marques, Abhilash Patra, Subrata Jana, et al. Bandgap of two-dimensional materials: Thorough assessment of modern exchange–correlation functionals. *The Journal of Chemical Physics*, 155(10), 2021.

[44] Chris E Mohn, Helmer Fjellvåg, Ponniah Vajeeston, Martin Valldor, and Kristin Bergum. Benchmarking density functional theory for accurate calculation of nitride band gaps. *Journal of Chemical Theory and Computation*, 2026.

[45] Mitsuaki Kawamura, Yuma Hizume, and Taisuke Ozaki. Benchmark of density functional theory for superconductors in elemental materials. *Physical Review B*, 101(13):134511, 2020.

[46] R Chitra and S Yashonath. Estimation of error in the diffusion coefficient from molecular dynamics simulations. *The Journal of Physical Chemistry B*, 101(27):5437–5445, 1997.

[47] Xingfeng He, Yizhou Zhu, Alexander Epstein, and Yifei Mo. Statistical variances of diffusional properties from ab initio molecular dynamics simulations. *npj Computational Materials*, 4(1):18, 2018.

[48] Yunsheng Liu, Xingfeng He, and Yifei Mo. Discrepancies and error evaluation metrics for machine learning interatomic potentials. *npj Computational Materials*, 9(1):174, 2023.