PREDICTION-ENHANCED MONTE CARLO: A MACHINE LEARNING VIEW ON CONTROL VARIATE

FENGPEI LI*[1]‡, HAOXIAN CHEN*[1]†, JIAHE LIN*[1], ARKIN GUPTA[1], XIAOWEI TAN[1], HONGLEI ZHAO[1], GANG XU[1], YURIY NEVMYVAKA[1], AGOSTINO CAPPONI[2], AND HENRY LAM[2]

Abstract. For many complex simulation tasks spanning areas such as healthcare, engineering, and finance, Monte Carlo (MC) methods are invaluable due to their unbiased estimates and precise error quantification. Nevertheless, MC simulations often become computationally prohibitive, especially for nested, multi-level, or path-dependent evaluations lacking effective variance reduction techniques. While machine learning (ML) surrogates appear as natural alternatives, naïve replacements typically introduce unquantifiable biases. We address this challenge by introducing Prediction-Enhanced Monte Carlo (PEMC), a framework that leverages modern ML models as *learned predictors*, using cheap and parallelizable simulations as *features*, to output unbiased evaluations with reduced variance and runtime. As a result, PEMC eliminates the closed-form-mean requirement that constrains classical control-variate methods, preserves unbiasedness and explicit confidence intervals of MC, and achieves scheme-wide variance reduction. Our theoretical analysis quantifies the optimal allocation between expensive evaluations and large batches of cheap, parallelizable feature draws. Across three representative applications—variance-swap pricing under stochastic-local-volatility, swaption pricing under Heath–Jarrow–Morton models, and ambulance diversion policy evaluation—we show that PEMC reduces root-mean-squared error by 30–55% relative to standard MC at similar computational cost.

Key words: Monte Carlo methods, Exotic Options Pricing, Control Variates, Variance Swaps, Swaptions.

1. Introduction

Monte Carlo (MC) is a fundamental tool for evaluating complex stochastic models where closed-form solutions are unavailable. Its main strengths—unbiasedness and rigorous uncertainty quantification through variance estimation and confidence intervals—make it essential in high-stakes, high-complexity applications such as healthcare resource allocation and financial risk management, where fairness or regulatory compliance are crucial. For example, transparent and bias-aware methods in healthcare are vital to ensure equitable outcomes and prevent disparities in care access [113, 77, 42]. Similarly, financial risk assessments rely on MC estimation to support hedging, re-balancing decisions, and valid error bounds, as minor biases in financial simulations can accumulate [30].

Despite these advantages, classical MC faces significant challenges in nested, multilevel, or path-dependent simulations that are difficult to parallelize. Complex stochastic dynamics can make each sample path extremely costly to generate—sometimes taking hours for one sample (e.g., to estimate credit valuation adjustments in over-the-counter contracts, see [63])—and effective variance reduction methods are often unavailable [9]. Consequently, MC's inherent $\mathcal{O}(1/\sqrt{n})$ convergence rate often prohibits real-time decision-making with adequate precision. This is the case, for example, in the context of pricing of exotic options, where sample paths need to be generated sequentially under complex dynamics, making parallelization not easily achievable [61]. These computational challenges

Date: June 10, 2025.

*Equal Contribution. [1]Morgan Stanley. [2]Department of IEOR, Columbia University. †The work was done during empolyment at Morgan Stanley.

‡Corresponding author: fl2412@columbia.edu.

are critical in financial derivative pricing, as highlighted in the literature, for example, in [74, Chapter 21] and [30, Chapter 4]. Similar computational bottlenecks arise in healthcare simulation, where modeling complex patient flows and hospital operations can require extensive computation time but transparent and bias-aware methods are crucial to ensure equitable outcomes [113, 77, 42].

Conversely, machine learning (ML) based predictive modeling has emerged as a fast alternative for approximating MC evaluations, including pricing financial derivatives [75, 53, 15]. Neural networks and other ML techniques can learn complex mappings and, once trained, can generate predictions at negligible marginal cost - what practitioners often call "fast evaluation" [73]. However, their black-box nature and lack of inherent statistical control hinder the quantification of errors, rendering them unsuitable for tasks requiring rigorous reliability assessments, such as high-stakes applications in risk management [75, 32]. Thus, MC and ML represent opposite ends of the spectrum: MC is reliable and uncertainty quantifiable but intrinsically slow, while ML is computationally fast but falls short in statistical guarantees.

Given this trade-off between the speed of ML and the reliability of MC, a natural question arises: can we design a method that combines both advantages while avoiding their respective pitfalls? In this paper, we provide a positive answer by introducing the *Prediction-Enhanced Monte Carlo (PEMC)* framework. PEMC leverages predictive ML modeling to replace a large portion of costly simulations with cheap/parallelizable samples from ML models, reducing variance on a scheme-wide level. Moreover, despite relying on black-box predictive models, PEMC maintains unbiasedness, balancing computational savings with rigorous statistical guarantees.

1.1. **Our Contributions.** PEMC innovates in two aspects. First, PEMC integrates ML predictive models into any MC baseline while preserving unbiasedness and error quantification of MC. Specifically, we show how the predictive modeling of ML combined with cheap, parallelizable samples will result in a reduction of variance and costly samples, enhancing real-time efficiency for complex MC simulations. Second, PEMC can also be viewed as a modernized variant of the control variate (CV) technique, a classical approach that reduces MC variance via auxiliary outputs whose known mean offsets random fluctuations [61, 9, 98]. Unfortunately, the classical CV approach, which requires a closed-form expression for the mean of a correlated auxiliary output, often becomes unavailable in many modern applications or complex stochastic systems, effectively barring the use of CV.

To explain concretely the second point above, our PEMC framework bridges ML with MC by substantially enlarging the scope of CV to many ML candidates. The key idea is the following: In the traditional CV framework, the known mean of CV is used to achieve a per-replication variance reduction while maintaining unbiasedness, i.e., the variance is reduced even if we run just one simulation replication. While this is a powerful feature, it comes at the cost of significantly limiting CV's applicability, due to the requirement of a closed-form expression for the mean. However, if the CV itself is computationally cheap in the sense of being estimatable with fast simulation, then we can afford to run plenty of additional MC *separately* from the original MC to estimate its mean. This would preserve unbiasedness and overall estimation variance relative to the total computation cost, as long as we consider efficiency improvement at the *scheme-wide* level. That is, in the PEMC framework, we consider total cost-aware instead of per-replication variance reduction, through which we relax the crucial requirement of known CV mean, and this in turn allows us to substantially expand the CV methodology to cover modern ML models. The alignment of our framework with the use of ML predictions prompts us to regard PEMC as a modernized view of CV.

1.2. **Literature Review.** The literature on enhancing MC simulations with ML approximators is extensive, yet many existing ML-based methods for CV and variance reduction impose restrictive conditions that appear to limit their practical applicability. Approaches such as reproducing Stein kernels [102, 89], Neural CV derived from Stein's identity [132, 83], regularized least squares for CV construction [109, 124, 86], adaptive CV schemes [67, 68, 80], and $\mathscr{L}^2$ function approximation

frameworks [94] have demonstrated theoretical promise with potentially rapid convergence rates. Recent theoretical work by [18] provides important insights into when regression-adjusted CVs can achieve optimal performance, showing that their effectiveness depends critically on function smoothness. However, these methods typically require knowledge of the CV mean, in turn restricting the classes of usable CV, or impose specific structural assumptions that can be difficult to check. As a result, while these existing methods offer strong theoretical foundations, their applicability in complex modeling scenarios can be undermined.

In a different line of literature, the works of [50] on quasi CVs and [104] on CVs using Estimated Means (CVEMs) is related to our approach, because they rely on an estimated mean (rather than a priori known, and thus not a traditional CV) of a quantity that is expected to exhibit a strong correlation with the random variable of interest. There are noticeable differences between our methodology and theirs. To begin with, our primary objective is to integrate modern ML capabilities into MC, so that our estimates can retain the error-quantifiability of the latter approach while leveraging the strengths of ML. To this end, a core component of our methodology regards what is needed, and how to train a suitable CV, rather than relying on a prefixed CV candidate, which may not always be obvious or even available. Moreover, our investigation includes the use of a pre-training approach in place of the adaptive algorithms employed in previous works, and provides a framework applicable to both finite-sample and asymptotic regimes, unlike the exclusively asymptotic focus of prior studies.

Within the literature of MC variance reduction, PEMC also relates to Multilevel Monte Carlo (MLMC) methods, originally introduced and popularized by [59], and also the idea of multi-fidelity modeling popular in scientific computation [106]. MLMC reduces variance by leveraging a hierarchy of discretizations (e.g., coarser and finer time steps in SDE simulations) and coupling them to achieve computational efficiency over naive Monte Carlo approaches [58]. The key idea is to couple simulations across resolution levels so that their differences exhibit reduced variance—or even enable unbiased estimators [115, 19, 118]. Along a similar vein, in multi-fidelity modeling, computational simulation models of different levels of resolution, which in turn bring in different accuracies and computational demands, are jointly utilized and concatenated, with input model parameters treated as random variables that allow coupling between the model levels [106]. Note that, despite the strong theoretical foundations, MLMC can be difficult to implement in practice, since constructing effective couplings, ensuring numerical stability, and tuning parameters to realize the theoretical complexity benefits are nontrivial tasks [110]. PEMC adopts the coupling paradigm of MLMC and multi-fidelity modeling but innovates by using ML predictors as CV.

Outside the MC literature, our PEMC framework aligns most closely with the recently proposed Prediction-Powered Inference (PPI) approach [6]. PPI is a statistical framework that enables researchers to construct valid confidence intervals and p-values when combining a small dataset with gold-standard labels and a large dataset with ML predictions. The key challenge PPI addresses is how to leverage abundant but potentially biased ML predictions alongside scarce but trustworthy ground-truth data for statistical inference. Such PPI-type hybrid strategy is becoming increasingly important in evaluating generative models such as large language models (LLMs) [138, 22, 52], where sparse human annotations is combined with abundant ML-generated labels to facilitate accurate model performance evaluation even in low-label scenarios. Additionally, the PPI methodologies also intersect with a rich literature in causal inference, specifically in doubly robust estimators [10, 56, 116] and orthogonal statistical learning [36, 55]. Doubly robust methods construct additional robustness layers by pairing sophisticated predictive models with auxiliary estimators, ensuring that minor errors in one model component do not significantly bias the overall estimation. This philosophy closely mirrors the logic of CV, and by extension, the core principles of our PEMC framework. While PPI replaces expensive gold-standard labels with cheaper ML predictions and auto-evaluation, PEMC focuses on replacing expensive, path-dependent and non-parallelizable MC samples with cheap, efficiently simulated, and highly parallelizable MC samples.

Finally, our work is also related to a branch of literature in computational finance, which has focused on harnessing ML to enhance the speed and accuracy of derivatives pricing. By using techniques such as neural network surrogates, these methods can approximate pricing functions or the Greeks at a fraction of the computational cost of traditional numerical schemes [75, 32, 122, 117, 85]. Early work by **(author?)** [75], for instance, use neural networks to relax stringent modeling assumptions, while more recent approaches [32, 122, 117, 85] demonstrate that deep architectures can effectively handle high-dimensional and path-dependent pricing problems. However, these purely data-driven approximations typically lack the rigorous error controls afforded by more classical methods. In the context of option pricing, even though both MC and partial differential equation (PDE)-based methods are widely used, it is known that PDE methods generally cannot be used to price options (i.e., payoff depends on the price path). While special cases allow for PDE formulations through state-space augmentation or boundary condition adjustments, these remain as exceptions rather than the rule [119].

The rest of the paper is organized as follows. In Section 2, we motivate and introduce PEMC. In Section 3, we provide more details on our framework via a running example of Asian option pricing. In Section 4, we present theoretical results and their practical implications. In Section 5, we apply PEMC to production-grade exotic option pricing, including variance swaps pricing under stochastic (local) volatility models and the pricing of swaptions under the HJM framework, as well as an ambulance diversion problem. We conclude the paper in Section 6 with discussions on broader implications and potential directions for future research.

2. The Prediction-Enhanced Monte Carlo Framework

We provide an overview and motivational explanation of our PEMC framework. Suppose we want to produce an unbiased evaluation of the quantity

$$\mu(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{\theta}}[f_{\boldsymbol{\theta}}(\boldsymbol{Y})] \tag{1}$$

for arbitrary, potentially high-dimensional parameter $\boldsymbol{\theta} \in \boldsymbol{\Theta} \subseteq \mathbb{R}^{d_\theta}$, where Θ is the parameter space. Here, $\boldsymbol{Y} \in \mathbb{R}^{d_y}$ represents a high-dimensional random vector whose probability distribution is parameterized by $\mathbb{E}_{\boldsymbol{\theta}}$ and $f_{\boldsymbol{\theta}}$ is the evaluation function $\mathbb{R}^{d_y} \to \mathbb{R}$. In general, both $f_{\boldsymbol{\theta}}$ and $\mathbb{E}_{\boldsymbol{\theta}}$ can depend on $\boldsymbol{\theta}$. The goal is to produce an unbiased estimate of $\mu(\boldsymbol{\theta})$ for any $\boldsymbol{\theta} \in \boldsymbol{\Theta}$, and the primary bottleneck stems from costly simulation of $\boldsymbol{Y}$.

2.1. Existing Challenges of MC, ML, and CV.

When $\boldsymbol{Y}$ can be simulated and $f_{\boldsymbol{\theta}}$ evaluated, then standard MC amounts to generating many $f_{\boldsymbol{\theta}}(\boldsymbol{Y}_i)$, $i = 1, \ldots, n$ and outputting their average. This approach is unbiased and its statistical error can be quantified straightforwardly via standard confidence intervals. However, it is demanding when evaluating $f_{\boldsymbol{\theta}}$ and simulating $\boldsymbol{Y}$ is expensive.

To reduce computation, one can approximate $\mu(\boldsymbol{\theta})$ with a ML regression model, say $\hat{\mu}(\boldsymbol{\theta})$, which can be evaluated more cheaply than simulating many $f_{\boldsymbol{\theta}}(\boldsymbol{Y}_i)$'s. This natural approach, however, would face challenges at multiple levels when reliability and error quantification is important: From a theoretical standpoint, ML generalization bounds typically assess training errors only within the model class, i.e., $\hat{\mu}(\boldsymbol{\theta}) - \mu^*(\boldsymbol{\theta})$ where $\mu^*(\cdot)$ is the best in-class model that can be different from $\mu(\cdot)$. Moreover, even assuming $\mu^*(\cdot)$ is a "rich" class, these bounds still only assess errors in an aggregate, worst-case fashion, i.e., high-probability bounds treating $\boldsymbol{\theta}$ as a random covariate. On the practical front, training-testing split or cross-validation succumbs to similar issues, in that it cannot measure bias against $\mu(\cdot)$, nor at the level of each $\boldsymbol{\theta}$. That is, in terms of reliability and uncertainty handling, MC appears superior by providing accurate, straightforward and instance-specific error assessment on each estimate of $\mu(\boldsymbol{\theta})$. None of these advantages are available for ML despite its computational edge.

An idea to leverage the respective advantages of MC and ML is to use CV which, as mentioned in Section 1, is a classical variance reduction method. CV aims to reduce the error of each MC run,

consequently the required replication size, by utilizing information from "auxiliary" simulation outputs. Specifically, in generating $f_{\boldsymbol{\theta}}(\boldsymbol{Y}_i)$, suppose we can also generate auxiliary output $\boldsymbol{X}_i$ and evaluate $g(\boldsymbol{\theta}, \boldsymbol{X}_i)$ for some function g. Then we can use $f_{\boldsymbol{\theta}}(\boldsymbol{Y}_i) - g(\boldsymbol{\theta}, \boldsymbol{X}_i) + \mathbb{E}[g(\boldsymbol{\theta}, \boldsymbol{X})]$ as an unbiased replicate for $\mathbb{E}_{\boldsymbol{\theta}}[f_{\boldsymbol{\theta}}(\boldsymbol{Y})]$. In particular, if $f_{\boldsymbol{\theta}}(\boldsymbol{Y})$ and $g(\boldsymbol{\theta}, \boldsymbol{X})$ are highly correlated, this replicate will have a smaller variance than $f_{\boldsymbol{\theta}}(\boldsymbol{Y}_i)$.

Naturally, one can think of using ML to train a good predictor $g(\boldsymbol{\theta}, \boldsymbol{X})$ for $f_{\boldsymbol{\theta}}(\boldsymbol{Y})$, which would be highly correlated with $f_{\boldsymbol{\theta}}(\boldsymbol{Y})$ and serve as an effective CV. However, taking aside the question of how to construct $\boldsymbol{X}$ and g, the CV method requires knowledge of $\mathbb{E}_{\boldsymbol{\theta}}[g(\boldsymbol{\theta}, \boldsymbol{X})]$, the closed-from mean on the auxiliary output. This knowledge is a key requirement for CV to achieve variance reduction. Yet, it is seldom available and has apparently severely limited the historical applicability of CV [62] and arguably even the more recent ML-inspired approaches [132, 124, 89].

2.2. **Our Remedy and Requirements.** Our PEMC aims to leverage the CV idea, with a twist to circumvent the need on the closed-form mean knowledge in order to allow for effective integration of MC with modern ML. This requires crucially a scheme-wide cost-aware view on variance reduction. Concretely, PEMC aims to estimate (1) by

$$PEMC(\boldsymbol{\theta}) := \frac{1}{n}\sum_{i=1}^{n}\left(f_{\boldsymbol{\theta}}(\boldsymbol{Y}_i) - g(\boldsymbol{\theta}, \boldsymbol{X}_i)\right) + \frac{1}{N}\sum_{j=1}^{N} g(\boldsymbol{\theta}, \tilde{\boldsymbol{X}}_j). \tag{2}$$

Here, $\boldsymbol{X}$ is an auxiliary variable that is dependent on $\boldsymbol{Y}$. Suppose, for now, that we are given the g function and the $\boldsymbol{X}$ variable. In equation (2), each pair of $(\boldsymbol{X}_i, \boldsymbol{Y}_i)$ is coupled and simulated together. On the other hand, each $\tilde{\boldsymbol{X}}_j$ is simulated independently from all $(\boldsymbol{X}_i, \boldsymbol{Y}_i)$'s, and has the same marginal distribution as $\boldsymbol{X}$. In this case, it is straightforward to see that (2) is an unbiased estimator of (1). On the other hand, in terms of cost efficiency, the total simulation cost consists of

$$[\text{cost of generating each } f_{\boldsymbol{\theta}}(\boldsymbol{Y}_i) - g(\boldsymbol{\theta}, \boldsymbol{X}_i)] \times n + [\text{cost of generating each } g(\boldsymbol{\theta}, \tilde{\boldsymbol{X}}_j)] \times N \tag{3}$$

while the total variance is

$$\frac{Var(f_{\boldsymbol{\theta}}(\boldsymbol{Y}_i) - g(\boldsymbol{\theta}, \boldsymbol{X}_i))}{n} + \frac{Var(g(\boldsymbol{\theta}, \tilde{\boldsymbol{X}}_j))}{N} \tag{4}$$

Suppose that g and $\boldsymbol{X}$ are designed such that the cost of generating each $g(\boldsymbol{\theta}, \tilde{\boldsymbol{X}}_j)$ is relatively negligible compared to each original MC target replicate $f_{\boldsymbol{\theta}}(\boldsymbol{Y}_i)$. Then, the total cost (3) will become approximately [cost of evaluating each $f_{\boldsymbol{\theta}}(\boldsymbol{Y}_i) - g(\boldsymbol{\theta}, \boldsymbol{X}_i)] \times n$, while the total variance (4) will become approximately $Var(f_{\boldsymbol{\theta}}(\boldsymbol{Y}_i) - g(\boldsymbol{\theta}, \boldsymbol{X}_i))/n$. In this case, the cost-variance structure is the same as the conventional CV method: If $g(\boldsymbol{\theta}, \boldsymbol{X})$ is a good predictor for $f_{\boldsymbol{\theta}}(\boldsymbol{Y})$, then $Var(f_{\boldsymbol{\theta}}(\boldsymbol{Y}_i) - g(\boldsymbol{\theta}, \boldsymbol{X}_i))$ is smaller than $Var(f_{\boldsymbol{\theta}}(\boldsymbol{Y}_i))$ and thus we attain a variance reduction to the overall estimator. Like in the conventional CV method, to achieve a total reduction from $Var(f_{\boldsymbol{\theta}}(\boldsymbol{Y}_i))/n$ to $Var(f_{\boldsymbol{\theta}}(\boldsymbol{Y}_i) - g(\boldsymbol{\theta}, \boldsymbol{X}_i))/n$, we generate n samples of $f_{\boldsymbol{\theta}}(\boldsymbol{Y}_i) - g(\boldsymbol{\theta}, \boldsymbol{X}_i)$, i.e., variance reduction is achieved per replication.

While the above depicts the idealized situation where we assume away the cost of generating $g(\boldsymbol{\theta}, \tilde{\boldsymbol{X}}_j)$, it provides guidance on what we need practically to elicit a cost-variance structure in (2), as revealed by (3) and (4), that is more favorable than naive MC. First, we need $g(\boldsymbol{\theta}, \boldsymbol{X})$ to serve as a good predictor for $f_{\boldsymbol{\theta}}(\boldsymbol{Y})$, but also that $\boldsymbol{X}$ has a marginal distribution that is both efficiently simulatable and highly parallelizable, allowing a large number N of independent samples of $\tilde{\boldsymbol{X}}_j$ with the same marginal as $\boldsymbol{X}$ to be generated at low cost. To this end, the selection of an appropriate $\boldsymbol{X}$ is a pivotal component of PEMC, analogous to feature selection in common ML tasks but with the additional need of being an efficiently simulatable, parallelizable random variable. Finding a good $\boldsymbol{X}$ often requires domain knowledge. In particular, given the whole path $\boldsymbol{Y}$, one can choose $\boldsymbol{X} := \phi(\boldsymbol{Y})$ to be some low-dimensional transformation ϕ of $\boldsymbol{Y}$ (e.g., ingredients for constructing $\boldsymbol{Y}$). For instance, for stochastic diferential equations (SDEs), one can define $\boldsymbol{X}$ as the sum of driving Brownian increments

that generate $\boldsymbol{Y}$. This choice could capture a substantial portion of the SDE's variance, making $\boldsymbol{X}$ a good feature for predicting $f(\boldsymbol{Y})$, while its Gaussianity allows it to be generated cheaply and in parallel.

Second, we require a function g to convert $\boldsymbol{\theta}$ and $\boldsymbol{X}$ into a good predictor for $f_{\boldsymbol{\theta}}(\boldsymbol{Y})$. In this work, g is pre-trained using a squared-error loss, with MC sample drawn under $(\boldsymbol{X}, \boldsymbol{Y}) \sim \mathbb{P}_{\boldsymbol{\theta}}$ for various $\boldsymbol{\theta} \sim \boldsymbol{\Theta}$, which aims to obtain a function $g(\boldsymbol{\theta}, \boldsymbol{X}) \approx \mathbb{E}_{\boldsymbol{\theta}}\big[f_{\boldsymbol{\theta}}(\boldsymbol{Y})\big|\boldsymbol{\theta}, \boldsymbol{X}\big]$. Note that this pre-training can be resource-consuming and takes significant amount of efforts. However, once this pre-training is done, the function g and our approach can be used to improve the efficiency in evaluating $\mu(\boldsymbol{\theta})$ for any $\boldsymbol{\theta}$. For example, in the context of option pricing, the pre-trained g can be stored as part of a model library tailored to a specific class of exotic options, enabling PEMC to be applied directly whenever new parameter configurations arise.

3. An Illustrative Case Study: Pricing Asian Options with PEMC

In this section, we employ Asian option pricing as a running example to explain the detailed steps and their implications in PEMC. Asian options are path-dependent derivatives whose payoffs depend on the average price of the underlying asset over a specified period. As mentioned in [23] and [45], closed-form solutions for pricing arithmetic average Asian options are generally not available. However, when the asset price dynamic follows a geometric Brownian motion (GBM), there exists an excellent CV candidate, using the geometric average instead of the arithmetic average, whose mean is precisely known via the Black-Scholes formula. This latter case often serves as the "textbook" example to demonstrate the power of CV [61]. As an important message, our case study here illustrates how PEMC can achieve a comparable level of improvements as the "textbook" CV, for evaluating arithmetic average Asian option price when the dynamics does *not* follow GBM. In other words, PEMC is capable to extend the scope of classical CV to potentially a much wider range of problems.

3.1. Problem Setup.

The quantity of interest is an evaluation that can be expressed via:

$$\text{Option Price}(\boldsymbol{\theta}) = \mathbb{E}_{\text{risk neutral measure}(\boldsymbol{\theta})}[f_{\text{payoff}(\boldsymbol{\theta})}(\boldsymbol{Y})], \tag{5}$$

where the risk neural measure is a probability measure where all financial securities are valued as if investors are indifferent to risk, and thus all assets are assumed to earn the risk-free rate [40].[1] For Asian option, we can specify the framework as:

(1) **Model Parameters $\boldsymbol{\theta}_{\text{model}}$**: These are the parameters that specify the simulation or the stochastic model of the underlying process $\boldsymbol{Y}$ (e.g., asset value time series). For instance, in a Heston model [70], the price process S_t and the instantaneous volatility process ν_t are jointly modeled:

$$\begin{aligned} dS_t &= rS_t dt + \sqrt{\nu}_t S_t dW_t^S \\ d\nu_t &= \kappa(\eta - \nu_t)dt + \delta\sqrt{\nu_t} dW_t^\nu \end{aligned} \tag{6}$$

The model parameter $\boldsymbol{\theta}_{\text{model}}$ is thus 5 dimensional $\boldsymbol{\theta}_{\text{model}} := (r, \eta, \delta, \rho, \kappa) \in \mathbb{R}^5$. It specifies the risk neutral drift r, the long-run average variance η, the volatility of volatility δ, the correlation ρ of between W^S and W^ν, and the mean reversion rate κ. In options pricing, these parameters are typically calibrated to fit market data and then used to generate sample paths $\boldsymbol{Y} := \{S_t\}_t$. Thus $\boldsymbol{\theta}_{\text{model}}$ governs and describes $\mathbb{E}_{\text{ risk neutral measure}}$ that generates $\boldsymbol{Y}$.

(2) **Simulation Parameters $\boldsymbol{\theta}_{\text{simulation}}$**: These parameters also specify $\mathbb{E}_{\text{risk neutral measure}}$. The distinction between $\boldsymbol{\theta}_{\text{simulation}}$ and $\boldsymbol{\theta}_{\text{model}}$ is somewhat artificial, primarily stemming from the convention that not all simulation parameters are calibrated externally, but rather serve as hyperparameters of the MC simulation itself (either customized or directly observed). For

[1] Under this measure, prices of financial securities are the discounted expected values of their future payoffs.

example, when one simulates the Heston model above using Euler discretization scheme,[2] we use $\boldsymbol{\theta}_{\text{simulation}}$ to specify the initial stock price S_0, the initial volatility ν_0, the time horizon T, and discretization time step Δt. The choice of these parameters can impact both the accuracy and efficiency of the simulation. While not typically part of the model calibration process, these parameters also play a crucial role in describing risk-neutral measure.

(3) **Payoff Function Parameters $\boldsymbol{\theta}_{\text{payoff}}$**: This parameter $\boldsymbol{\theta}_{\text{payoff}}$ parametrizes the *payoff function* $f_{\boldsymbol{\theta}_{\text{payoff}}}$. In option pricing, this specifies the details of the contract. For example, given the full price path $\boldsymbol{Y} := \{S_t\}_t$, the payoff function for an arithmetic Asian options takes the form

$$P_A(\{S_t\}_t) = \max\left(\frac{1}{n_D}\sum_{i=1}^{n_D} S_{t_i} - K, 0\right) \tag{7}$$

where $\boldsymbol{\theta}_{\text{payoff}}$ specifies the strike level K, the sampling frequency n_D and the observation dates $\{t_i\}_{i\in[n_D]}$. Notice we have simply omitted the discount factor e^{-rT}. Note that the discrete sampling defined by the payoff does not necessarily determine the Euler discretization in time and in fact the latter is usually much more frequent for accuracy.

Having specified the distinct roles of $\boldsymbol{\theta}_{\text{model}}$, $\boldsymbol{\theta}_{\text{simulation}}$ and $\boldsymbol{\theta}_{\text{payoff}}$, we will henceforth refer to them collectively as $\boldsymbol{\theta}$ when no ambiguity arises.

3.2. **Feature Engineering to Design X.** We now follow the general ideas laid out in Section 2.2, starting with the creation of the variable $\boldsymbol{X}$ to couple with $\boldsymbol{Y}$. As described in Section 2.2, we want to find some input feature $\boldsymbol{X} := \phi(\boldsymbol{Y})$ that is some low-dimensional transformation ϕ of $\boldsymbol{Y}$ with:

- Property 1: $\boldsymbol{X}$ can be regarded as a reasonable predictor for $f_{\boldsymbol{\theta}_{\text{payoff}}}(\boldsymbol{Y})$.
- Property 2: In contrast to that of $\boldsymbol{Y}$, the marginal distribution of $\boldsymbol{X}$ allows for efficient and parallelizable simulation.

To this end, we take $\boldsymbol{X}$ as the sum of two correlated Brownian increments driving (6), motivated by two key properties:

- Predictability: The sum of Brownian increments explains away a portion of the variance of the SDE.
- Marginal simulation: The sum of two Brownian increments follows a correlated Gaussian distribution, which can be generated efficiently.

More precisely, when sampling $\boldsymbol{Y}$, we collect $\boldsymbol{X}$ to be a two-dimensional vector consisting of

$$W_T^S := \sum_{j=1}^{T/\Delta t} \Delta W_{j\Delta t}^S \quad \text{and} \quad W_T^\nu := \sum_{j=1}^{T/\Delta t} \Delta W_{j\Delta t}^\nu,$$

simulated during each step of the Euler scheme with discretization scale Δt. The marginal of $\boldsymbol{X}$ is simply a two-dimensional Gaussian: $\frac{\boldsymbol{X}}{\sqrt{T}} \sim \mathcal{N}\left(\mathbf{0}, \left[\begin{smallmatrix} 1 & \rho \\ \rho & 1 \end{smallmatrix}\right]\right)$.

Using Brownian increments, while natural, is not the only choice. As with most ML tasks, feature engineering that leverages domain expertise can yield better results. We illustrate with another example, the pricing of a floating strike lookback call option, under the Heston model (6). Here, the option payoff is given by:

$$f(\{S_t\}_t) = \max(S_T - \min_{0\le t\le T} S_t, 0)$$

[2]As mentioned in [45], Euler scheme was the primary method for simulation of Heston model until the exact simulation of [31] is developed. However, due to its significant computational demands, the exact method is less commonly used in practice and simpler, biased simulation method such as Quadratic Exponential [3] or Euler scheme remains prevalent (see, e.g., [92]).

where S_t is the asset price process and T is the option maturity. A good choice of feature could be:

$$\boldsymbol{X} = (W_T^S, \min_{0\le t\le T}(r-\eta/2)t + \sqrt{\eta}W_t^S).$$

The rationale behind the construction of $\min_{0\le t\le T}(r-\eta/2)t+\sqrt{\eta}W_t^S$ term is grounded in the domain knowledge of the Heston model. Specifically, since η represents the mean-reverting level of variance process ν_t, we may approximate $\nu_t \approx \eta$ (i.e., assume fixed volatility) and simplify the asset price process to a GBM:

$$S_t \approx S_0 e^{(r-\eta/2)t+\sqrt{\eta}W_t^S} \text{ and } \min_{0\le t\le T} S_t \approx S_0 e^{\min_{0\le t\le T}(r-\eta/2)t+\sqrt{\eta}W_t^S}.$$

This approximation leads to the expression $f(\{S_t\}_t) \approx S_0 e^{(r-\nu/2)T+\sqrt{\nu}[\boldsymbol{X}]_1} - S_0 e^{[\boldsymbol{X}]_2}$, which could serve as a potentially effective approximation of $f(\{S_t\}_t)$. This corresponds directly to Property 1 in the construction of $\boldsymbol{X}$ in 3.3. For Property 2, the marginal distribution of $\boldsymbol{X}$ permits efficient and parallelizable simulation. In particular, applying Girsanov's Theorem enables one to derive a closed-form expression for the joint density of $([\boldsymbol{X}]_1, [\boldsymbol{X}]_2)$ [79]. This result makes it straightforward to sample these components directly.

3.3. **Building Prediction Model.** We now focus on the pre-training phase of our prediction model. The data generation and model training process can be outlined through the following key steps:

(1) We first define a parameter space $\boldsymbol{\Theta}$ that encompasses all combinations of $\boldsymbol{\theta} := (\boldsymbol{\theta}_{\text{model}}, \boldsymbol{\theta}_{\text{simulation}}, \boldsymbol{\theta}_{\text{payoff}})$ that are of practical interest. This space covers the range of parameters that would likely be encountered in real-world pricing scenarios on a daily, weekly or monthly basis, depending on the model update frequency.
(2) We then draw samples of $\boldsymbol{\theta}_i$ from $\boldsymbol{\Theta}$, using uniform sampling or any measure that ensures comprehensive coverage. For each sampled parameter set, we generate one pair [3] of $\boldsymbol{Y}(\boldsymbol{\theta}_i)$ and its corresponding $\boldsymbol{X}(\boldsymbol{\theta}_i) := \phi(\boldsymbol{Y}(\boldsymbol{\theta}_i))$. It is important to note that $\boldsymbol{X}$ and $\boldsymbol{Y}$ are coupled in this generation process.
(3) For each generated pair, we compute the payoff $f_{\boldsymbol{\theta}_{i,\text{payoff}}}(\boldsymbol{Y}(\boldsymbol{\theta}_i))$, which serves as a *label* in our training data. The corresponding *feature* vector is comprised of $(\boldsymbol{\theta}_{i,\text{model}}, \boldsymbol{\theta}_{i,\text{simulation}}, \boldsymbol{\theta}_{i,\text{payoff}}, \boldsymbol{X}(\boldsymbol{\theta}_i))$.
(4) This process is repeated N_{train} times to generate training dataset which consists of N_{train} pairs of

$$\begin{aligned} \textit{feature}_i :=& (\boldsymbol{\theta}_{i,\text{model}}, \boldsymbol{\theta}_{i,\text{simulation}}, \boldsymbol{\theta}_{i,\text{payoff}}, \boldsymbol{X}(\boldsymbol{\theta}_i)) \\ \textit{label}_i :=& f_{\boldsymbol{\theta}_{i,\text{payoff}}}(\boldsymbol{Y}(\boldsymbol{\theta}_i)) \end{aligned}$$

for $i \in [N_{\text{train}}]$.
(5) The ML model g is then trained to minimize the MSE loss and saved:

$$\min_g \frac{1}{N_{\text{train}}} \sum_{i\in[N_{\text{train}}]} (\text{label}_i - g(\text{feature}_i))^2. \tag{8}$$

The essential steps are summarized in Algorithm 1. While the data generation process can be time-intensive due to the complexity of $\boldsymbol{Y}$, it is conducted offline during the pre-training phase. This approach minimizes its impact on real-time pricing applications by effectively trading memory usage for computational speed during execution.

This pre-training setup presents an ideal scenario for ML applications. It is important to also note that PEMC does not require g to take specific forms. However, the convex and differentiable nature of the loss function (i.e., squared loss), the large supply of data (i.e., from Monte Carlo simulation), and the need for expressiveness of the model family make neural networks (NNs) an ideal choice. For training NNs, various optimization techniques and architectural enhancements—such as

[3] You could also generate multiple pair of $(\boldsymbol{Y}(\boldsymbol{\theta}_i), \boldsymbol{X}(\boldsymbol{\theta}_i))$ under the same sampled $\boldsymbol{\theta}_i$.

Algorithm 1 Prediction Model Training in PEMC

procedure DATAGENERATION($N_{\text{train}}, \boldsymbol{\Theta}$)
  Initialize empty datasets (*features*, *labels*)
  for $i = 1$ **to** N_{train} **do**
    Sample parameters $\boldsymbol{\theta}_i \sim \Theta$ uniformly
    Generate $\boldsymbol{X}(\boldsymbol{\theta}_i), \boldsymbol{Y}(\boldsymbol{\theta}_i) \sim \mathbb{E}_{\text{risk neutral measure}(\boldsymbol{\theta}_i)}$
    $label_i \leftarrow f_{\boldsymbol{\theta}_{i,\text{payoff}}}(\boldsymbol{Y}(\boldsymbol{\theta}_i))$
    $feature_i \leftarrow (\boldsymbol{\theta}_{i,\text{model}}, \boldsymbol{\theta}_{i,\text{simulation}}, \boldsymbol{\theta}_{i,\text{payoff}}, \boldsymbol{X}(\boldsymbol{\theta}_i))$
    Store $(feature_i, label_i)$ to (*features*, *labels*)
  end for
  return datasets (*features*, *labels*)
end procedure

procedure TRAINING(*features*, *labels*)
  Initialize model g
  $N_{\text{train}} \leftarrow \text{length}(features)$
  Minimize: $\min_g \frac{1}{N_{\text{train}}} \sum_{i \in [N_{\text{train}}]} (\text{label}_i - g(\text{feature}_i))^2$
  return trained model g
end procedure

the Adam optimizer [81], batch normalization [76], and skip connections [65]—can be employed to improve performance. In fact, recent advancements in ML technologies have made the training process remarkably accessible and efficient. The availability of open-source, user-friendly frameworks like PyTorch [105], TensorFlow [1], and JAX [24]—coupled with GPU acceleration and highly optimized C++ backends—has dramatically simplified the process of training models on large datasets. These frameworks support recent neural network architectures, such as Convolutional Neural Networks (CNNs) [84], Residual Networks (ResNets) [65], and Transformers [131], as well as various Stochastic Gradient Descent (SGD) optimization algorithms [21]. In applications, we only utilized well-established and widely recognized NN structures that are now considered common. These models can typically be implemented using just a few lines of code, as they have been incorporated into existing packages and extensively optimized.

3.4. **Evaluation.** Once we finish the pre-training phase, the prediction model is stored and used on-the-fly in the evaluation process as follows:

(1) A specific parameter $\boldsymbol{\theta} := (\boldsymbol{\theta}_{\text{model}}, \boldsymbol{\theta}_{\text{simulation}}, \boldsymbol{\theta}_{\text{payoff}}) \in \boldsymbol{\Theta}$ is given, e.g., calibrated from market data in real time, and needs to be used for pricing.
(2) Generate n pairs of $(\boldsymbol{Y}_i, \boldsymbol{X}_i)_{i \in [n]}$, from $\mathbb{E}_{\text{risk neutral measure}(\boldsymbol{\theta})}$. Here we have suppressed $\boldsymbol{Y}$'s dependency on $\boldsymbol{\theta}$ for convenience.
(3) Based on the marginal distribution of $\boldsymbol{X}$, independently generate N additional samples of $(\tilde{\boldsymbol{X}}_j)_{j \in [N]}$ from $\mathbb{E}_{\text{risk neutral measure}(\boldsymbol{\theta})}$, directly according to its marginal distribution. These samples are independent of the previous data $(\boldsymbol{Y}_i, \boldsymbol{X}_i)_{i \in [n]}$.
(4) Utilize the pre-trained model g to evaluate the PEMC estimator:

$$PEMC := \frac{1}{n} \sum_{i=1}^{n} (f(\boldsymbol{Y}_i) - g(\boldsymbol{X}_i)) + \frac{1}{N} \sum_{j=1}^{N} g(\tilde{\boldsymbol{X}}_j) \tag{9}$$

For notational simplicity, we have written $g(\boldsymbol{X})$ instead of $g(\boldsymbol{\theta}_{\text{model}}, \boldsymbol{\theta}_{\text{simulation}}, \boldsymbol{\theta}_{\text{payoff}}, \boldsymbol{X})$, and $f(\boldsymbol{Y})$ instead of $f_{\boldsymbol{\theta}_{\text{payoff}}}(\boldsymbol{Y})$. A key aspect of this procedure is the relationship between N and n. Typically,

we choose N to be one, or several orders of magnitude larger than n. The optimal choice of N versus n, will be discussed Section 4. The PEMC evaluation step is summarized in Algorithm 2.

Algorithm 2 Evaluation Procedure in PEMC

1: **procedure** EVALUATION($\boldsymbol{\theta}, n, N, g$)
2: Generate i.i.d. $(\boldsymbol{Y}_i, \boldsymbol{X}_i)_{i\in[n]}$ from $\mathbb{E}_{\text{risk neutral measure}(\boldsymbol{\theta})}$
3: Generate i.i.d. $(\tilde{\boldsymbol{X}}_j)_{j\in[N]}$ independently from its marginal in $\mathbb{E}_{\text{risk neutral measure}(\boldsymbol{\theta})}$
4: Compute $PEMC := \frac{1}{n}\sum_{i=1}^{n}(f(\boldsymbol{Y}_i) - g(\boldsymbol{X}_i)) + \frac{1}{N}\sum_{j=1}^{N} g(\tilde{\boldsymbol{X}}_j)$
5: **return** $PEMC$
6: **end procedure**

3.5. **Overall Methodology.** We now summarize our overall PEMC methodology applied to the Asian option under the Heston model, i.e., (7), with $\boldsymbol{\theta}_{\text{payoff}}$ specifying the strike level K, the sampling frequency n_D and the observation dates $\{t_i\}_{i\in[n_D]}$. The Heston model is described in (6) which requires $\boldsymbol{\theta}_{\text{model}} := (r, \eta, \delta, \rho, \kappa) \in \mathbb{R}^5$. Suppose we use an NN as the prediction model. Then, the PEMC framework could proceed as follows:

(1) Define the parameter space $\boldsymbol{\Theta}$ that encompasses realizations of

$$\boldsymbol{\theta} := (\underbrace{r, \eta, \delta, \rho, \kappa}_{\boldsymbol{\theta}_{\text{model}}}, \underbrace{S_0, \nu_0, \Delta t, T}_{\boldsymbol{\theta}_{\text{simulation}}}, \underbrace{K, n_D, \{t_i\}_{i\in[n_D]}}_{\boldsymbol{\theta}_{\text{payoff}}})$$

which are of practical interest.
(2) Uniformly sample $\boldsymbol{\theta}$ from $\boldsymbol{\Theta}$. This step is straightforward if the parameter space is a Cartesian product of intervals. For each sampled $\boldsymbol{\theta}$, generate process $\boldsymbol{Y} := (S_t, \nu_t)_t$ using a discretization scheme with step size Δt, with the Heston model specified by $\boldsymbol{\theta}$.
(3) During the sampling of $\boldsymbol{Y}$, we collect $\boldsymbol{X}$ to be the sum of Brownian increment

$$W_T^S := \sum_{j=1}^{T/\Delta t} \Delta W_{j\Delta t}^S \quad \text{and} \quad W_T^\nu := \sum_{j=1}^{T/\Delta t} \Delta W_{j\Delta t}^\nu,$$

simulated during each step of the Euler scheme. This makes the marginal of $\boldsymbol{X}$ simply a two-dimensional Gaussian: $\frac{\boldsymbol{X}}{\sqrt{T}} \sim \mathcal{N}\Big(\mathbf{0}, \left[\begin{smallmatrix} 1 & \rho \\ \rho & 1 \end{smallmatrix}\right]\Big)$.
(4) Save

$$\begin{aligned} \textit{label} :=& (\frac{1}{n_D}\sum_{i=1}^{n_D} S_{t_i} - K)^+ \\ \textit{feature} :=& (r, \eta, \delta, \rho, \kappa, S_0, \nu_0, \Delta t, T, K, n_D, \{t_i\}_{i\in[n_D]}, W_T^S, W_T^\nu). \end{aligned} \tag{10}$$

(5) Repeat steps 2-4 N_{train} times to generate dataset $(\textit{feature}_i, \textit{label}_i)_{i\in[N_{\text{train}}]}$ of size N_{train}.
(6) Train a NN with weights $\boldsymbol{w}$ to minimize the MSE loss:

$$\min_{\boldsymbol{w}} \frac{1}{N_{\text{train}}} \sum_{i\in[N_{\text{train}}]} (\text{label}_i - NN_{\boldsymbol{w}}(\text{feature}_i))^2$$

and use $\hat{w}$ to approximate

$$NN_{\hat{\boldsymbol{w}}} \approx \mathbb{E}\left[(\frac{1}{n_D}\sum_{i=1}^{n_D} S_{t_i} - K)^+ \middle| r, \eta, \delta, \rho, \kappa, S_0, \nu_0, \Delta t, T, K, n_D, \{t_i\}_{i\in[n_D]}, W_T^S, W_T^\nu\right].$$

(7) At evaluation, given a specific $\boldsymbol{\theta} = (r, \eta, \delta, \rho, \kappa, S_0, \nu_0, \Delta t, T, K, n_D, \{t_i\}_{i\in[n_D]}, W_T^S, W_T^\nu) \in \Theta$, we generate n paired samples $(label_i, feature_i)_{i\in[n]}$ as in (10). We also generate N i.i.d. $\sqrt{T}\mathcal{N}\left(\mathbf{0}, \left[\begin{smallmatrix} 1 & \rho \\ \rho & 1 \end{smallmatrix}\right]\right)$ samples and store them as $(\tilde{feature})_{j\in[N]}$.

(8) Output $PEMC := \frac{1}{n}\sum_{i=1}^n (label_i - NN_{\hat{\boldsymbol{w}}}(feature_i)) + \frac{1}{N}\sum_{j=1}^N NN_{\hat{\boldsymbol{w}}}(\tilde{feature}_j)$

In Appendix B, we present numerical results on the performance comparisons of PEMC with standard MC and CV for Asian options under GBM. In this case, an excellent CV using the Black-Scholes formula is available under the standard CV framework for us to benchmark against. At the same time, in Section 5, we present results on more realistic and technically complex applications, including several production-grade exotic options that are vastly more complex than Asian options.

4. Analysis and Performance

In this section, we provide a detailed and comprehensive analysis of the PEMC estimator. All omitted proofs are delegated to Appendix A. All properties of PEMC discussed here pertain to the evaluation stage, where $\boldsymbol{\theta}$ is arbitrary but fixed. Henceforth, we omit the explicit dependence on $\boldsymbol{\theta}$ in the notation (e.g., writing $\boldsymbol{Y}$ in place of $\boldsymbol{Y}(\boldsymbol{\theta})$).

4.1. Bias Analysis.

Theorem 1 (Unbiasedness)**.** *The PEMC estimator in equation (9), is unbiased:*

$$\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^n (f(\boldsymbol{Y}_i) - g(\boldsymbol{X}_i)) + \frac{1}{N}\sum_{j=1}^N g(\tilde{\boldsymbol{X}}_j)\right] = \mathbb{E}[f(\boldsymbol{Y})]$$

Proof. Proof First note that g is pre-trained. The proof then follows since $\boldsymbol{X}_i$ and $\tilde{\boldsymbol{X}}_j$ have the same marginal distribution for any i, j. □

Given the unbiased nature of PEMC, the analysis of PEMC as a Monte Carlo method primarily hinges on its variances and computational costs.

4.2. Variance Analysis.

To facilitate the analysis, we first introduce some necessary notation.

Definition 2. *Given function f that takes input $\boldsymbol{Y}$ and g that takes input $\boldsymbol{X}$ (so both f, g are considered fixed here), we denote*

$$\sigma_{f-g}^2 := \mathrm{Var}(f(\boldsymbol{Y}) - g(\boldsymbol{X})),\ \sigma_f^2 := \mathrm{Var}(f(\boldsymbol{Y})),\ \text{and } \sigma_g^2 := \mathrm{Var}(g(\boldsymbol{X})).$$

Similarly, we denote cost of generating a coupled sample $f(\boldsymbol{Y}) - g(\boldsymbol{X})$ as c_{f-g}, the cost of generating $f(\boldsymbol{Y})$ as c_f and the cost of generating $g(\boldsymbol{X})$ as c_g.

Lemma 1. *Using the notations from 2, we have*

$$\mathrm{Cov}(f(\boldsymbol{Y}), g(\boldsymbol{X})) = \frac{\sigma_f^2 + \sigma_g^2 - \sigma_{f-g}^2}{2}.$$

We make two remarks here. First, since we generally obtain $\boldsymbol{X}$ in the same process during the generation of $\boldsymbol{Y}$, thus typically in either sense of the cost, we have $c_{f-g} \approx c_f$. Second, the term cost is intentionally left unspecified because its interpretation varies depending on the context. In some context, cost refers to the sample size, while in others it represents the wall-clock time required to generate and evaluate all the samples, making the concept of cost instance dependent. Therefore, for now, we use cost as an umbrella term to capture various meanings under different contexts.

Lemma 2. *The Variance of the PEMC estimator in equation (9) is*

$$\mathrm{Var}(PEMC) = \frac{1}{n}\sigma_{f-g}^2 + \frac{1}{N}\sigma_g^2.$$

Proof. Proof First note that g is pre-trained. Moreover, in the evaluation phase, data $(\boldsymbol{Y}_i, \boldsymbol{X}_i)_{i\in[n]}$ are generated independent of $(\tilde{\boldsymbol{X}}_j)_{j\in[N]}$. □

Having established the unbiasedness and the variance form of the PEMC estimator, we could turn to inference. In particular, using consistent estimates of variance, the application of the central limit theorem enables the construction of valid asymptotic confidence intervals. Moreover, a range of inferential techniques may be employed—such as deriving high-probability bounds or other non-asymptotic guarantees—the asymptotic confidence interval construction suffices to illustrate the point. The approach follows closely from results in prediction-powered inference [6].

Theorem 3 (Asymptotic Confidence Intervals)**.** *In the set up of the evaluation algorithm 2, given the prediction model g, $(\boldsymbol{Y}_i, \boldsymbol{X}_i)_{i\in[n]}$ and $(\tilde{\boldsymbol{X}}_j)_{j\in[N]}$, we define*

$$\hat{\sigma}^2_{f-g} = \frac{1}{n}\sum_{i=1}^{n}\Big(f(\boldsymbol{Y}_i) - g(\boldsymbol{X}_i) - \Big(\frac{1}{n}\sum_{i'=1}^{n} f(\boldsymbol{Y}_{i'}) - g(\boldsymbol{X}_{i'})\Big)\Big)^2,$$

and

$$\hat{\sigma}^2_{g} = \frac{1}{N}\sum_{j=1}^{N}\big(g(\tilde{\boldsymbol{X}}_j) - \frac{1}{N}\sum_{j'=1}^{N} g(\tilde{\boldsymbol{X}}_{j'})\big)^2,$$

be the respective sample variance. Let $z_{1-\alpha/2}$ denote the $(1-\alpha/2)$-quantile of the standard normal distribution. Then, the interval

$$\left(PEMC - z_{1-\alpha/2}\sqrt{\frac{\hat{\sigma}^2_{f-g}}{n} + \frac{\hat{\sigma}^2_{g}}{N}}, \quad PEMC + z_{1-\alpha/2}\sqrt{\frac{\hat{\sigma}^2_{f-g}}{n} + \frac{\hat{\sigma}^2_{g}}{N}}\right),$$

is an asymptotically valid $1-\alpha$ confidence interval for $\mathbb{E}[f(\boldsymbol{Y})]$, as $n, N \to \infty$.

Proof. Proof Theorem 1 establishes PEMC is unbiased. Moreover, $\hat{\sigma}^2_{f-g}, \hat{\sigma}^2_{g}$ are consistent estimates of $\sigma^2_{f-g}, \sigma^2_{g}$, the results follows from Lemma 2 and the central limit theorem and Slutsky's theorem. □

4.3. **Variance Reduction.** Having analyzed the bias and variance of the PEMC estimator, we now turn to a fundamental practical question: Under a fixed cost budget—whether measured in terms of the number of samples, computational time, or other limited resources—when does PEMC outperform standard Monte Carlo (MC)? Our aim is to understand how PEMC's variance reduction scales with its cost, thereby identifying regimes where PEMC emerges as more efficient than standard MC. To this end, we first determine the optimal allocation of the sample sizes n (the number of expensive full simulations) and N (the number of cheap, feature-only evaluations) within the PEMC framework, given a total resource budget B. This investigation provides a guideline for parameter selection and reveals conditions under which PEMC's gains are maximized.

Lemma 3. *Assuming $n, N \in \mathbb{R}^+$ and $c_g, c_{f-g} \in \mathbb{R}^+$, the optimal allocation between n and N for PEMC, for any positive budget B, follows as*

$$\frac{n}{N} = \frac{\sigma_{f-g}}{\sigma_g}\cdot\sqrt{\frac{c_g}{c_{f-g}}} \quad . \tag{11}$$

Proof. Proof By relaxing the constraints to $n, N \in \mathbb{R}^+$, the optimization problem

$$\begin{aligned}\min_{n>0,N>0} \quad & \frac{1}{n}\sigma^2_{f-g} + \frac{1}{N}\sigma^2_{g}.\\ \text{s.t.} \quad & nc_{f-g} + Nc_g \le B.\end{aligned}$$

is jointly convex in $n, N \in \mathbb{R}^+$. The objective value also approaches infinity as $N \to 0$ or $n \to 0$. Consequently, the strict convexity ensures the existence of a global minimizer in the interior and directly solving the Lagrangian gives us the result. □

In practice, $c_{f-g} \gg c_g$ often holds which suggests one should set $N \gg n$ according to Lemma 3. While Lemma 3 relies on treating $n, N \in \mathbb{R}^+$ as continuous—ignoring the integer constraints $n, N \in \mathbb{N}^+$— this continuous approximation still provides valuable guidance when $n, N \geq 1$. In fact, in practice, we found the ratio suggested by Lemma 3 guides near-optimal parameter selection. However, the standard MC corresponds to the case $N = 0$, a scenario not covered by Lemma 3, and thus requires separate consideration. Building on Lemma 3, we can estimate an upper bound on the variance reduction PEMC can achieve relative to standard MC under these idealized assumptions.

Lemma 4. *Assume $c_{f-g} = c_f$. In the same setup as Lemma 3, the variance ratio between PEMC under the optimal allocation and standard MC follows as*

$$\frac{\mathrm{Var}(PEMC)}{\mathrm{Var}(MC)} = \frac{\sigma_{f-g}^2}{\sigma_f^2}\left(1 + \frac{\sigma_g}{\sigma_{f-g}} \cdot \sqrt{\frac{c_g}{c_f}}\right) + \frac{\sigma_g^2}{\sigma_f^2}\left(\frac{\sigma_{f-g}}{\sigma_g} \cdot \sqrt{\frac{c_g}{c_f}} + \frac{c_g}{c_f}\right). \tag{12}$$

Proof. Proof Based on the results of Lemma 3, we can deduce that this ratio again does not depend on the budget B, so we omit its dependence in the statement. Moreover, we can conveniently choose a budget of the form $B = n_0 c_{f-g} + n_0 \frac{\sigma_g}{\sigma_{f-g}} \cdot \sqrt{\frac{c_{f-g}}{c_g}} c_g$ which gives an allocation of $n = n_0$ and $N = n_0 \frac{\sigma_g}{\sigma_{f-g}} \cdot \sqrt{\frac{c_{f-g}}{c_g}}$ for PEMC in accordance with Lemma 3, while comparing with B/c_f samples for standard MC. The rest follows from Lemma 2 and the assumption $c_f = c_{f-g}$. □

Based on the result of Lemma 4, a natural question arises: how can we gauge $\frac{\sigma_{f-g}^2}{\sigma_f^2}$ or $\frac{\sigma_g^2}{\sigma_f^2}$ in practice? How are these ratios tied to the quality of the predictive model g in PEMC? As we shall see, the extent of variance reduction that PEMC can deliver is linked to how well g is trained. To illustrate this, we first consider an ideal scenario where g is trained to the optimum, i.e., the true regression function.

Lemma 5. *Suppose $c_{f-g} = c_f$, $f(\boldsymbol{Y})$ is square-integrable and $g = \mathbb{E}[f(\boldsymbol{Y}) \,|\, \boldsymbol{X}]$. Define $\rho := \mathrm{corr}(f, g)$ and $c := \frac{c_g}{c_f}$. Then we have $\rho = \frac{\sigma_g}{\sigma_f}$ and the ratio $\frac{\mathrm{Var}(PEMC)}{\mathrm{Var}(MC)}$ in Lemma 4 reduces to*

$$r(\rho, c) := (1 - \rho^2)\left(1 + \frac{\rho}{\sqrt{1-\rho^2}} \cdot \sqrt{c}\right) + \rho^2\left(\frac{\sqrt{1-\rho^2}}{\rho} \cdot \sqrt{c} + c\right). \tag{13}$$

Proof. Proof When $g = \mathbb{E}[f(\boldsymbol{Y}) \,|\, \boldsymbol{X}]$ is the true regressor, i.e.,

$$g = \underset{h \text{ measurable}}{\arg\min} \ \mathbb{E}[(f(\boldsymbol{Y}) - h(\boldsymbol{X}))^2],$$

and f is square-integrable, it follows that $\mathbb{E}[(f-g)h] = 0$ for all square-integrable h. Plug in $h = g$, we obtain $\mathrm{Cov}(f - g, g) = 0$. Consequently we have $\rho := \mathrm{corr}(f, g) = \frac{\sigma_g}{\sigma_f}$ and $\sigma_f^2 = \sigma_{f-g}^2 + \sigma_g^2$, which further implies $\sigma_g = \rho^2 \sigma_f^2$ and $\sigma_{f-g}^2 = (1 - \rho^2)\sigma_f^2$. The rest follows from $c_{f-g} = c_f$. □

Lemma 5 says that the variance reduction of PEMC relative to standard MC can be approximated by the function $r(\rho, c)$ in (13), where $\rho = \sigma_g/\sigma_f$ captures the predictive power of $g(\boldsymbol{X})$ for $f(\boldsymbol{Y})$, and $c = c_g/c_f$ quantifies the relative cost of evaluating $g(\boldsymbol{X})$ versus generating full samples for $f(\boldsymbol{Y})$. In Figure 1, we visualize the variance reduction function $r(\rho, c)$ for $\rho, c \in (0, 1)$. On the left is a contour plot of $r(\rho, c)$ where the red line traces the curve $r(\rho, c) = 1$, corresponding to the "break-even" boundary where PEMC's variance matches that of MC. Regions below this line, $\{(\rho, r) \in [0, 1]^2 | r(\rho, c) < 1\}$, represent regimes of ρ and c where PEMC achieves variance reduction, i.e., where ρ is sufficiently large and c is sufficiently small. On the right, the graph of $r(\rho, c)$ as a function of ρ when $c = 0.001$ provides a clear benchmark: a correlation $\rho = 0.5$ yields approximately 22.2% variance reduction, while $\rho = 0.7$ yields about 45.8%. In our PEMC applications, we design feature $\boldsymbol{X}$ and predictive model g so that, for most $\boldsymbol{\theta} \in \Theta$ during evaluation, we can safely gauge that c falls between 10^{-2} to

10^{-3} and ρ exceeds 0.5. In these regions, Lemma 3 suggests choosing N/n in the range of 5 to 20 for near-optimal performance. In our experiments, a ratio of $N = 10n$ proved effective.

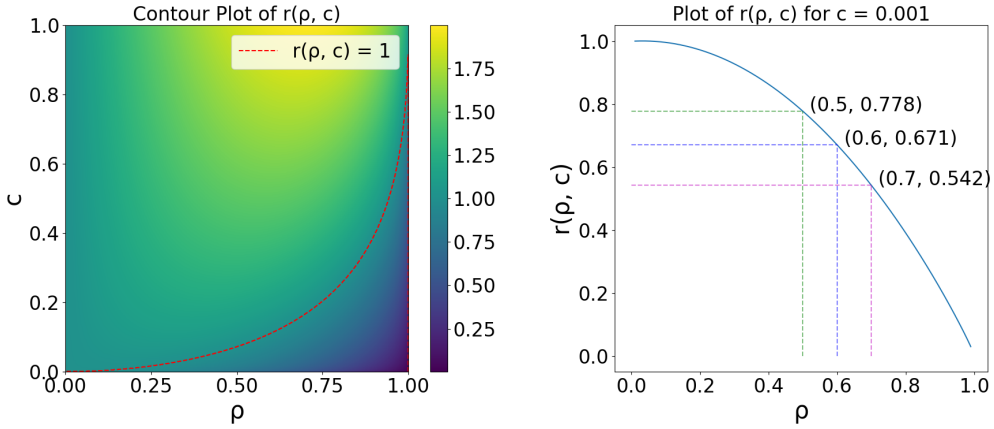

FIGURE 1. Variance Reduction Function $r(\rho, c)$. Left: A contour map of the variance reduction ratio $r(\rho, c)$ as a function of the correlation $\rho = \sigma_g/\sigma_f$ and relative cost $c = c_g/c_f$. The red curve indicates the level set $r(\rho, c) = 1$; the regimes where PEMC outperforms standard MC lie below the curve. Right: A graph of $r(\rho, c)$ as a function of ρ when $c = 10^{-3}$.

4.3.1. *Learning Theory Estimates.* The statements in Lemma 3 assume that g represents the true regression function $\mathbb{E}[f|\boldsymbol{X}]$. In practice, however, g is obtained through a learning procedure applied to finite training samples, and thus will only approximate the true conditional expectation. Consequently, any practical implementation of PEMC must account for the approximation and estimation errors inherent in the ML training process, and this error also varies with $\boldsymbol{\theta}$. To rigorously quantify these effects, one can invoke tools from statistical learning theory, such as uniform convergence bounds, VC-dimension, or Rademacher complexities [129, 13, 95]. These quantities, by relating the complexity of the function class used to represent g and the available training sample size N_{train}, lead to error bounds that ensure, with high probability, that the trained predictor g is close to the true regression function within a controlled margin. We avoid delving deeply into them so as not to distract from our main theme. Instead, we adopt a standard, off-the-shelf and 'placeholder'-type result, whose proof is detailed in the Appendix, that illustrates the typical form of guarantees one could expect from PEMC.

Lemma 6. *Under the regularity conditions specified in the Appendix, for any $\epsilon, \delta > 0$, there exists a sufficiently large sample size N_{train} and a suitably chosen neural network class from which we select and train a predictor g on N_{train} samples, such that*

$$\frac{\text{Var}(PEMC)}{\text{Var}(MC)} \leq r(\rho, c) + \epsilon$$

for a randomly evaluated $\theta \sim \Theta$ with probability at least $1 - \delta$.

4.4. **Control Variate Coefficient.** In this section we explore a connection between the PEMC framework and traditional Control Variate (CV) methods. In (9), note that we could introduce a free parameter a to create a variant:

$$PEMC(a) := \frac{1}{n}\sum_{i=1}^{n}(f(\boldsymbol{Y}_i) - ag(\boldsymbol{X}_i)) + \frac{1}{N}\sum_{j=1}^{N} ag(\tilde{\boldsymbol{X}}_j) \tag{14}$$

The introduction of a is analogous to CV approaches, where the parameter a does not introduce bias but can be tuned to minimize variance. In PEMC, this viewpoint may appear somewhat redundant since ag is also a valid prediction model, which suggests a would be chosen implicitly during the training of the prediction model. Nevertheless, suppose g is fixed. Then $\frac{1}{n}\sum_{i=1}^{n} -g(\boldsymbol{X}_i) + \frac{1}{N}\sum_{j=1}^{N} g(\tilde{\boldsymbol{X}}_j)$ can be viewed as a readily available, zero-mean control variate for $\frac{1}{n}\sum_{i=1}^{n} f(\boldsymbol{Y}_i)$. We can then choose a which minimizes the variance. The optimal a^* then is:

$$a^* = \frac{\mathrm{Cov}(f(\boldsymbol{Y}), g(\boldsymbol{X}))}{(n/N+1)\mathrm{Var}(g(\boldsymbol{X}))}.$$

In the ideal scenario, we have $N \gg n$ and $g(\boldsymbol{X}) = \mathbb{E}[f(\boldsymbol{Y})|\boldsymbol{X}]$, which would imply $\mathrm{Cov}(f(\boldsymbol{Y}) - g(\boldsymbol{X}), g(\boldsymbol{X})) = 0$ and $\mathrm{Cov}(f(\boldsymbol{Y}), g(\boldsymbol{X})) = \mathrm{Var}(g(\boldsymbol{X}))$. This leads to $a^* = \frac{1}{1+n/N} \approx 1$, the default choice of PEMC, i.e., $a = 1$.

5. Applications

After establishing the theoretical foundations of PEMC and demonstrating its effectiveness in controlled environments, we now apply it to real-world financial scenarios where the complexity of models often renders standard variance reduction techniques difficult or unavailable. In this section, we first examine PEMC in practice through two complex options pricing problems. In particular, we examine variance swaps pricing under stochastic local volatility models [47, 44, 57], and the pricing of swaptions under the Heath-Jarrow-Morton (HJM) framework [34]. In both cases, the path-dependent nature of the contracts and the complexity of the model dynamics make traditional control variates difficult.

5.1. Variance Swaps in stochastic local vol models.

Variance swaps are financial derivatives that enable investors to trade future realized volatility against current implied volatility [14]. Unlike traditional options, which derive their value from the underlying asset's price, variance swaps are based on the variance of the asset's returns over a specified period [43]. This unique structure allows for pure exposure to volatility, making variance swaps valuable tools for risk management and speculative strategies [57, 111]. In this subsection, we focus on using PEMC to price variance swaps under a stochastic local volatility (SLV) model [114], as closed-form formulas exist for variance swaps in stochastic volatility models such as Heston [137], but not for SLV [128] which provides greater flexibility through a data-intensive, non-parametric specification of the volatility surface [57]. SLV models emerged as a hybrid approach that combines the market-implied local volatility surface with stochastic volatility dynamics [64], providing practitioners with greater calibration flexibility and more accurate price reproduction across strike-maturity ranges [114].

Unlike the Heston model's parsimonious parameter set, SLV requires handling a full volatility surface discretized on a dense 2D grid, effectively making the parameter space $\boldsymbol{\Theta}$ for PEMC high-dimensional. To efficiently process this grid-structured volatility data, we adopt a Convolutional Neural Network (CNN) architecture [84] for PEMC, effectively feeding the local vol surface as an image into predictive modeling. The CNNs are particularly well-suited for this task as they naturally exploit the spatial relationships in the volatility surface, similar to their success in image processing tasks [65, 82].

Under the SLV model, we adopt the following SDEs for simulating the asset price $\{S_t\}_t$:

$$\mathrm{d}S_t = rS_t\mathrm{d}t + \sigma(S_t, t)e^{\nu_t}S_t\mathrm{d}W_t^S, \tag{15}$$

$$\mathrm{d}\nu_t = \kappa(\eta_t - \nu_t)\mathrm{d}t + \delta\mathrm{d}W_t^\nu, \tag{16}$$

with $\langle \mathrm{d}W_t^S, \mathrm{d}W_t^\nu \rangle = \rho dt$ and $\eta_t := -\frac{\delta^2}{2\kappa}(1 + e^{-2\kappa t})$. Here $\sigma(\cdot,\cdot) : \mathbb{R} \times \mathbb{R}^+ \to \mathbb{R}^+$ is 2D function representing the (interpolated) local volatility surface and $\exp(\nu_t)$ is a stochastic multiplier with $\exp(\nu_0) = 1$ and $\mathbb{E}[e^{2\nu_t}] = 1$. In practice, local volatility surfaces are calibrated to and stored as discrete two-dimensional grids indexed by asset prices (spot) and time. During simulation, these

discrete values are interpolated as needed to obtain volatilities at arbitrary price-time points [37]. For our PEMC implementation, we treat this discrete grid as part of our input parameter $\boldsymbol{\theta}_{\text{model}}$, reflecting how the market-calibrated surfaces would be used in practice. While local volatility models [47, 44] and their calibration to market data constitute an extensive research area in their own right, our focus here is on the PEMC implementation. Thus, we assume a calibrated local volatility has been given for evaluation, regardless of the method used to obtain it. Following practical conventions, we store $\sigma(\cdot,\cdot)$ on a $|\mathcal{S}| \times |\mathcal{T}|$ grid, where $\mathcal{S}$ contains $|\mathcal{S}|$ equally-spaced price points in $[S^{\min}_{\text{surface}}, S^{\max}_{\text{surface}}]$ and $\mathcal{T}$ contains $|\mathcal{T}|$ equally-spaced time points in $[t^{\min}_{\text{surface}}, t^{\max}_{\text{surface}}]$. At each point on the grid, we store the value of local volatility according to [34]:

$$\sigma^2_{base}(x,t) = \frac{\sum_{i=0}^{2} p_i \tau_i e^{-x^2/(2t\tau_i^2)-t\tau_i^2/8}}{\sum_{i=0}^{2}(p_i/\tau_i) e^{-x^2/(2t\tau_i^2)-t\tau_i^2/8}}, \quad \text{with } p_0 := 1-p_1-p_2,\ x := \log(S_t/S_0), \tag{17}$$

for $p_0 = 0.3, p_1 = 0.5, p_2 = 0.2, \tau_0 = 0.4, \tau_1 = 0.3, \tau_2 = 0.6$ as in Figure 2 in [34]. While this analytical form (17) is used in our data-generating process, it is important to note that PEMC treats the surface as any market-calibrated volatility surface - accessing it only through its discrete grid values. When sampling $\boldsymbol{\theta} \sim \boldsymbol{\Theta}$ to produce σ grid, we add a $\mathcal{N}(0,\xi^2)$ noise independently to all the $|\mathcal{S}| \times |\mathcal{T}|$ points in the grid, on top of their baseline value σ_{base} (17). A path is then generated by Euler's scheme:

$$\begin{aligned}
&\nu_{t+\Delta t} \leftarrow \nu_t + \kappa(\eta_t - \nu_t)\Delta t + \delta \Delta W^{\nu}_t, \\
&S_{t+\Delta t} \leftarrow S_t \exp\Big(\big(r - \tfrac{1}{2}\tilde{\nu}^2_{t+\Delta t}\big)\Delta t + \tilde{\nu}_{t+\Delta t}\Delta W^S_t\Big), \quad \text{where } \tilde{\nu}_{t+\Delta t} = \widehat{\sigma}(S_t,t)e^{\nu_{t+\Delta t}},
\end{aligned}$$

with $(\Delta W^S_t, \Delta W^{\nu}_t) \overset{\text{i.i.d}}{\sim} \sqrt{\Delta t}\cdot\mathcal{N}(\mathbf{0}, \left[\begin{smallmatrix}1 & \rho\\ \rho & 1\end{smallmatrix}\right])$, and $\widehat{\sigma}(S_t,t)$ obtained by interpolations of the grid σ. The input $(\boldsymbol{\theta}, \boldsymbol{X})$ can be represented as

$$\text{feature}_i = \{\underbrace{\{\sigma^2(s,t)\}_{s\in\mathcal{S},t\in\mathcal{T}}}_{\text{surface info}}, \underbrace{r,\delta,\kappa,\rho,\mu}_{\boldsymbol{\theta}_{\text{model}}}, \underbrace{S_0,\nu_0}_{\boldsymbol{\theta}_{\text{simulation}}}, \underbrace{K}_{\boldsymbol{\theta}_{\text{payoff}}}, \underbrace{(W^S_T, W^{\nu}_T)}_{\mathbf{X}(\boldsymbol{\theta})}\}.$$

The parameters space $\boldsymbol{\Theta}$, as well as the evaluation $\boldsymbol{\theta}$, is summarized in Table 1.

TABLE 1. Parameter Setup in SLV Model

mode	ξ	S_0	r	κ	λ	ρ	T, Δt
$N_{\text{train}} = 3,000,000$	0.02	$[50, 150]$	0.02	$[1.5, 4.5]$	$[0.1, 1.0]$	$[-0.2, -0.9]$	1, 1/252
evaluation	0	100	0.02	3.0	0.5	-0.5	

To handle the 2D grid of high-dimensional volatility surface data, we design a two-branch neural network architecture, which is illustrated in Figure 2. The first branch processes the discretized volatility surface $\sigma^2(x,t)$ using a CNN architecture inspired by VGG [121], which has become a standard choice for image processing tasks and is readily available in modern deep learning packages like PyTorch [105]. This branch then consists of two 2D convolutional layers interspersed with ReLU activations, followed by a MaxPool2d operation for dimensionality reduction. The surface features are then flattened through a fully connected layer to produce an embedding. The second branch handles the remaining model parameters in $(\boldsymbol{\theta}, \boldsymbol{X})$ through a series of fully connected layers with dropout regularization, batch normalization, and ReLU activations, ultimately producing another embedding. Finally, the two separate embeddings are then fed into a "Synthesizer" module, which combines the information through additional fully connected layers with dropout and ReLU activations to produce the final prediction. This architecture choice is motivated by the proven effectiveness of CNNs in handling grid-structured data [84, 65], and particularly the VGG architecture's success in extracting hierarchical features while maintaining relative simplicity [121]. The details of NN architecture are summarized in Table 12.

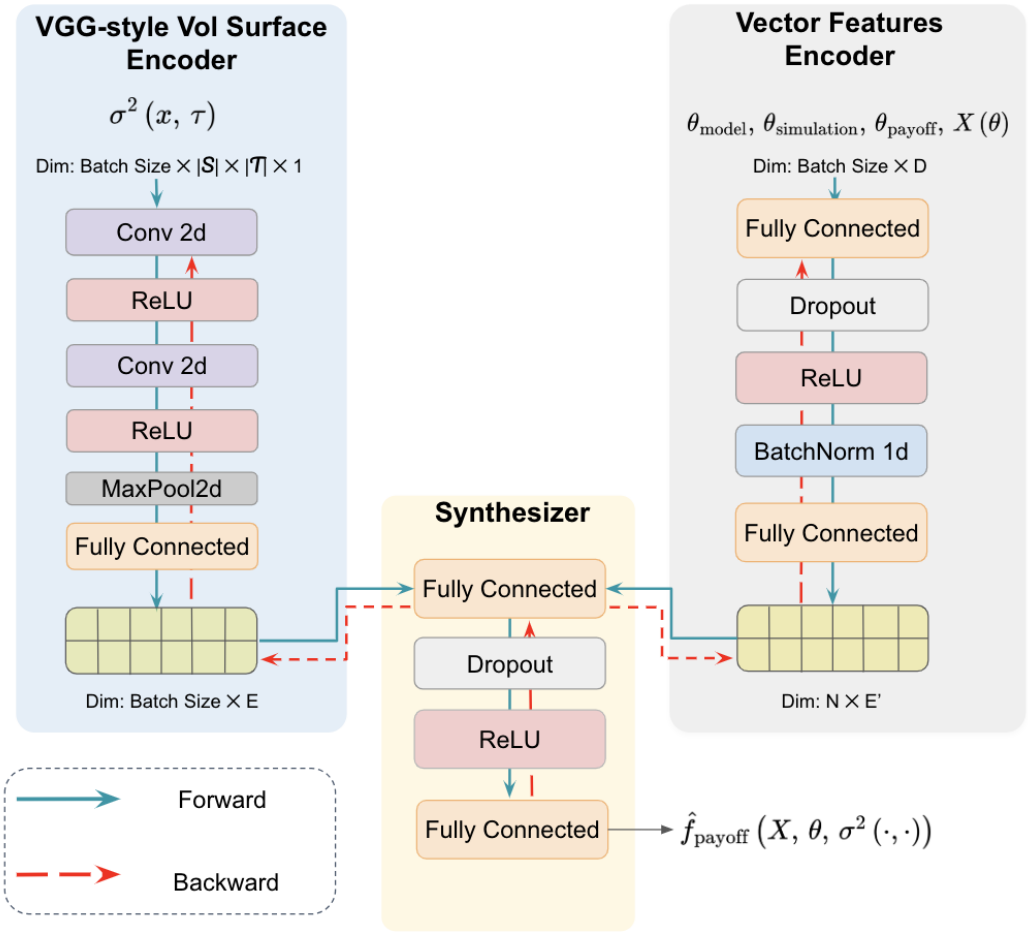

FIGURE 2. Neural Network architecture for the SLV Model.

TABLE 2. Neural Network Architecture Parameters

CNN Branch	**Feed-forward Branch**
kernel size: 3, stride: 1, padding: 1	hidden dim: 512
max pooling (kernel: 2, stride: 2, padding: 0)	output dim: 128

Finally, we evaluate PEMC's performance in the SLV setting across sample sizes $n \in \{1000, 2000, 4000, 8000, 10000, 20000\}$ with $N = 10n$, benchmarking against a ground truth computed from 5×10^7 MC samples. The results, based on 200 independent experiments and shown in Table 3, reveal that PEMC's effectiveness persists even in this more complex setting. Despite the added complexity of handling high-dimensional volatility surfaces, PEMC achieves a 30-40% reduction in mean squared error compared to standard MC, demonstrating its robustness as a variance reduction technique across different model frameworks.

5.2. **Swaptions in HJM Models.** Interest rate derivatives play a crucial role in financial markets, with swaptions being particularly important instruments for managing interest rate risk [29]. A swaption gives its holder the right to enter into an interest rate swap at a future date, providing

TABLE 3. RMSE from 200 Experiments for Pricing Variance Swap under the SLV Model

Method	$n = 1000$	$n = 2000$	$n = 4000$	$n = 6000$	$n = 8000$	$n = 10000$	$n = 20000$
Monte Carlo (MC)	0.0206	0.0145	0.0101	0.0075	0.0064	0.0065	0.0047
PEMC	0.0130	0.0088	0.0061	0.0055	0.0040	0.0040	0.0027

flexibility in hedging future interest rate exposures [74]. The pricing of these instruments, however, presents significant computational challenges due to the high-dimensional nature of interest rate modeling [5]. In this subsection, we demonstrate PEMC's application to swaption pricing under the Heath-Jarrow-Morton (HJM) framework [66]. The HJM model directly describes the evolution of the entire forward rate curve, offering greater flexibility than traditional short-rate models. For illustration purposes, we focus on a one-factor specification with exponential volatility structure [61], though the framework readily extends to multi-factor cases. A swaption is a contract granting its holder the right, but not the obligation, to enter into an interest rate swap at a future date. In a standard interest rate swap, one party agrees to pay a fixed rate while receiving a floating rate, and the other party does the opposite. Consider a swap with n_p fixed payment periods, each of length $\Delta t'$, starting at time t'_0 and ending at time $t'_{n_p} = t'_0 + \sum_{l=1}^{n_p} \Delta t'$. The value of this swap at time t'_0 is:

$$V_{t'_0} = C\left(R\sum_{l=1}^{n_p} B(t'_0, t'_l)\Delta t' + B(t'_0, t'_{n_p}) - 1\right),$$

where C is the notional amount (contract size), R is the fixed rate, and $B(t'_0, t'_l)$ is the discount factor from t'_0 to $t'_l F$. A swaption provides the holder with the option to enter into this swap at t'_0. The payoff of the swaption is simply

$$\max(0, V_{t'_0}),$$

and its expectation under the risk-neutral measure gives the price of swaptions. To specify the risk neutral measure, one needs to model the bond price. The price of a zero-coupon bond $B(t,T)$ maturing at time T is given by the forward rate $f(t,u)$ as: $B(t,T) = \exp\left(-\int_t^T f(t,u)\,du\right)$, or equivalently $\frac{\partial \log B(t,T)}{\partial T} = -f(t,T)$. The HJM framework [61] then models the dynamics of forward rate curve directly:

$$df(t,T) = \mu(t,T)\,dt + \sigma(t,T)^\top dW(t),$$

where $\mu(t,T)$ is the drift, $\sigma(t,T)$ is the volatility function of the forward rate, and $W(t)$ is a Brownian motion. In contrast to short-rate models (e.g., Vasicek [130] or Cox-Ingersoll-Ross [39]), which only model the dynamics of the short-term interest rate, the HJM model directly models the dynamics of the entire term structure of interest rates [66]. The HJM model is widely used in practice because of its flexibility in modeling interest rate derivatives like swaptions and its ability to incorporate complex volatility structures [29, 5]. However, the model's generality also leads to the need for sophisticated numerical methods for simulation [61]. A key property of the HJM model is the no-arbitrage condition [61], which specifies the drift completely by the volatility:

$$\mu(t,T) = \sigma(t,T)^\top \int_t^T \sigma(t,u)\,du. \tag{18}$$

Thus, in the HJM framework, the model is fully specified by defining the initial forward rate curve $f(0,T)$ and the structure of the volatility $\sigma(t,T)$. In our experiment we used a simple one factor HJM for illustration.

5.2.1. *PEMC for HJM.* Just as with the local volatility surface case, in practice $\sigma(t,T)$ cannot be predefined parametrically and must be calibrated from market data of caps, floors, and swaptions, yielding a discrete grid of values. However, for demonstration purposes, we employ a classical exponential decay specification as our baseline model inspired from [61]:

$$\sigma_{base}(t,T) = \sigma_0 \exp(-\alpha_\sigma(T-t)) \tag{19}$$

with σ_0 and α_σ as part of $\boldsymbol{\theta}$. Similarly, for a baseline initial forward curve, we use:

$$f_{base}(0,T) = f_0 + c_f(1-\exp(-\alpha_f T)). \tag{20}$$

with f_0, c_f, and α_f part of the $\boldsymbol{\theta}$. While this analytical form serves as our data-generating process, PEMC accesses it only through its discrete grid values with added noise. This approach mirrors our treatment of the local volatility surface in the previous section, where we used the parametric form in [34] solely as a realistic baseline for generating training data.

Indeed, while HJM and these analytical forms (19)-(20) are formulated in continuous time, in practice we need to implement numerical discretization. Following the scheme in [61], one discretizes both the time axis and the set of maturities i.e., time steps $\mathcal{T}_t = \{t_0, t_1, \ldots, t_{N_T}\}$ and maturity points $\mathcal{T} = \{t_1, t_2, \ldots, t_{N_M}\}$ forming a time-maturity grid. In our experiment, for simplicity, we assume they share one grid $\mathcal{T}$ of size $|\mathcal{T}|$. Then, when sampling $\boldsymbol{\theta} \sim \boldsymbol{\Theta}$, one first sample $\sigma_0, \alpha_\sigma, f_0, c_f, \alpha_f$, then one sample the grids $\{\sigma(t_i,t_j)\}_{t_i \le t_j, t_i, t_j \in \mathcal{T}}$ and $\{f(0,t_i)\}_{t_i \in \mathcal{T}}$ from (19) and (20) with added noise $\mathcal{N}(0, (\frac{\sigma_0}{2(t_j+5)})^2)$ and $\mathcal{N}(0, (\frac{1}{100(t_j+5)})^2)$ respectively, on top of their baseline values. The noise level $\xi = \frac{\sigma_0}{2(t+5)}$. Paths are then generated using the discretization scheme described in [61] (e.g., $f(t_i,t_j) = f(t_{i-1},t_j) + \mu(t_{i-1},t_j)(t_i - t_{i-1}) + \sqrt{t_i - t_{i-1}}\sigma(t_{i-1},t_j)\mathcal{N}(0,1)$ where $\mu(t_{i-1},t_j)$ is the discretized drift term determined through (25)). We refer interested readers to [61] for the complete derivation and implementation details of this standard simulation scheme.

For PEMC, our the parameter $\boldsymbol{\theta} \sim \boldsymbol{\Theta}$ includes the volatility parameters $(\sigma_0, \alpha_\sigma)$ in (19) and initial forward curve parameters (f_0, c, α_f) in (20). The swaption's fixed rate is set as $R = \exp(-\frac{\sum_i^{n-1} f(0,t_i')}{T_{\text{final}} - t_0'})$, reflecting the common practice where swap rates are typically determined in reference to the prevailing forward rate curve rather than being arbitrarily chosen. Here the parameters are notional amount C, start time t_0', payment interval $\Delta t'$, and number of payments n_p. The simulation uses time step Δt up to final maturity T_{final}. During training, parameters are sampled uniformly from the ranges specified in Table 4 where the evaluation $\boldsymbol{\theta}$ is also listed.

TABLE 4. Parameter Setup in HJM Model

Mode	σ_0	α_σ	f_0	c_f	α_f	R	C	t_0'	$\Delta t'$	n_p	Δt	T^*
$N_{\text{train}} = 3,000,000$	$[0.01, 0.03]$	$[0.001, 0.9]$	$[0.01, 0.03]$	$[0.01, 0.05]$	$[0.001, 0.9]$	$\exp(-\frac{\sum_{i=0}^{n-1} f(0,t_i')}{T^* - t_0'})$	100	5	1	20	1/52	25
evaluation	0.0015	100	0.02	3.0	0.5							

The input for PEMC is, similar as before:

$$\text{feature} = \{\underbrace{\{\sigma(t_i,t_j)\}_{t_i \le t_j, t_i, t_j \in \mathcal{T}}}_{\text{volatility structure}}, \underbrace{\sigma_0, \alpha_\sigma, f_0, c_f, \alpha_f}_{\boldsymbol{\theta}_{\text{model}}}, \underbrace{\{f(0,t_i)\}_{t_i \in \mathcal{T}}}_{\text{initial forward curve}}, \underbrace{(W_T^S, W_T^\nu)}_{\mathbf{X}}\}.$$

To handle both the two-dimensional grid of volatility structure $\sigma(t,T)$ and the one dimensional grid of initial forward curve $f(0,T)$, we design a three-branch neural network architecture, illustrated in Figure 3. The first branch, labeled"2D Function Encoder", processes the 2d volatility structure grid using a CNN architecture with two 2D convolutional layers, each followed by batch normalization. The branch concludes with an average pooling operation and produces an embedding. The second branch processes the initial forward curve $f(0,T)$ grid through a "1D Function Encoder" utilizing 1D convolutional layers - a natural choice for sequential data [84] - followed by batch normalization and average pooling to produce another embedding. The third branch handles the remaining input

through fully connected layers with batch normalization. Finally, these three separate embeddings are then fed into a “Synthesizer" module that combines the information through multiple fully connected layers with batch normalization, ultimately producing the final prediction. This architecture effectively leverages both the spatial structure of the volatility surface through 2D CNNs [121], the sequential nature of the forward curve through 1D CNNs [103], and the scalar parameters through standard deep learning techniques [65]. The complete network architecture is detailed in Table 13.

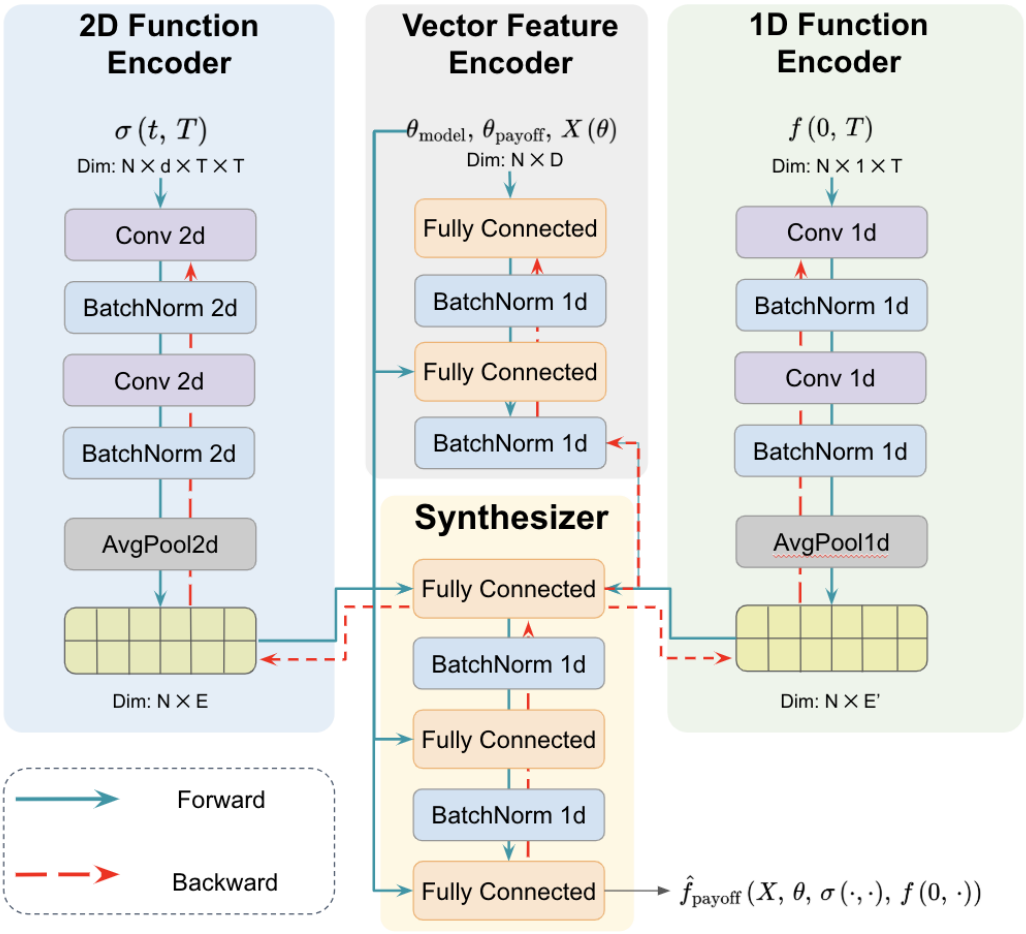

FIGURE 3. Neural network architecture for modeling Swaption payoff.

Finally, following the same evaluation methodology, we assess PEMC's performance in the HJM swaption pricing setting across sample sizes $n \in \{1000, 3000, 5000, 7000, 9000, 11000\}$ with $N = 10n$, using 5×10^7 MC samples to establish the ground truth value. As presented in Table 6, the outcomes from 200 independent experiments confirm that PEMC remains highly effective within the context of interest rate derivatives. Despite the added complexity of managing both volatility structures and forward rate curves, PEMC achieves a 45-50% reduction in RMSE to standard Monte Carlo methods. This performance aligns with the variance reduction levels observed in our earlier examples. The boxplot is shown in Figure 4. The consistent effectiveness of PEMC across various financial instruments and modeling frameworks further underscores its versatility as a robust variance reduction technique.

2D function branch parameters	**1D function branch parameters**	**Vector feature branch parameters**
Kernel size: (1, 3)	Kernel size: 10	
Stride: (1, 3)	Stride: 3	
Padding: 0	Padding: 0	Hidden dim: 512
AvgPool2d kernel size: (2, 2)	AvgPool1d kernel size: 2	Output dim: 128
AvgPool2d stride: (2, 2)	AvgPool1d stride: 2	
AvgPool2d padding: 0	AvgPool1d padding: 0	
Feed-forward Synthesizer Parameters		
Hidden dim: 128		
Output dim: 1		

TABLE 5. Hyper-parameter setup for the neural network

TABLE 6. Root Mean Squared Error from 200 Experiments

Method	$n = 1000$	$n = 3000$	$n = 5000$	$n = 7000$	$n = 9000$	$n = 11000$
Monte Carlo (MC)	0.0096	0.0062	0.0048	0.0039	0.0035	0.0029
PEMC	0.0055	0.0028	0.0024	0.0019	0.0018	0.0015

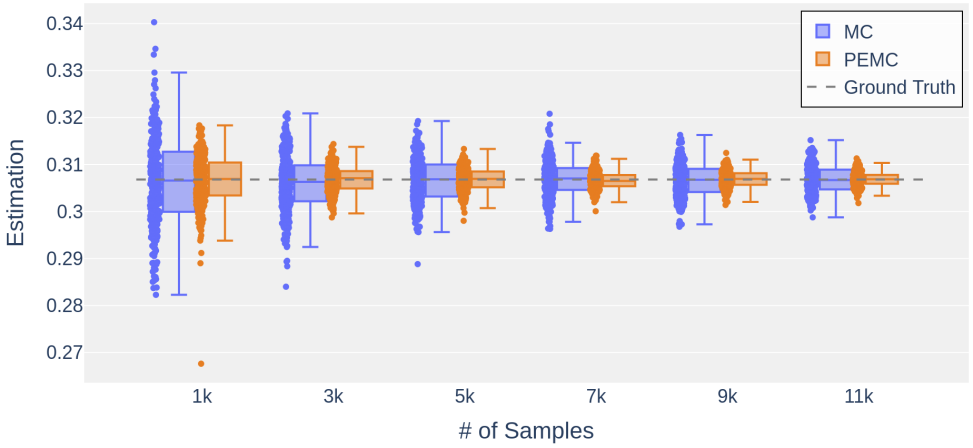

FIGURE 4. Boxplots for HJM Experiments.

5.3. **Discussions, Extensions and More Examples.** We conclude with several observations and potential extensions of the PEMC framework.

5.3.1. *Evaluation Metric.* First, while training the neural network estimator g using mean squared error (MSE) loss is common, it is not always clear how to interpret the resulting MSE score. Unlike some well-established benchmarks (e.g., classification accuracy), there is no canonical threshold or known "good" MSE value for a given problem. This ambiguity makes it challenging to determine when the network is sufficiently trained. To address this, we can exploit the fact that g is meant to represent the conditional expectation $g = \mathbb{E}[f \mid \text{input}]$. If the network approximates this expectation

well, then the sample average of $g(\boldsymbol{X})$ should be close to the sample average of $f(\boldsymbol{Y})$ over a given dataset. One practical diagnostic is to compute the Mean Absolute Relative Error (MARE) between these two averages. If $\mathbb{E}[g(\boldsymbol{X})] \approx \mathbb{E}[f(\boldsymbol{Y})]$, it provides a tangible indication that g is capturing the underlying expectation. Our empirical experience suggests this criterion is very effective in practice (a 5-1% MARE typically indicates exceptional PEMC), complementing common techniques like early stopping in machine learning workflows. More importantly, as the we have shown in the theory, if g closely approximates the conditional expectation, the variance reduction in PEMC is guaranteed—even if marginally—relative to standard Monte Carlo.

5.3.2. *XVA, Greeks and Quasi-Monte Carlo.* Beyond pricing exotic options, this approach naturally extends to other computationally intensive Monte Carlo settings in quantitative finance. Notably, adjustments like Credit valuation adjustment (CVA) and more complex XVAs often require vast simulation runs, sometimes taking days to complete [63, 4]. Integrating PEMC with a well-trained neural network can cut these computational times drastically while preserving accuracy.

Moreover, in the pricing of exotic derivatives and other path-dependent instruments, automatic differentiation (AD) techniques are often used to compute Greeks efficiently. By leveraging modern deep learning frameworks—such as PyTorch or TensorFlow—that support automatic differentiation natively, we can easily differentiate the trained network g with respect to input parameters [60]. This enables quick and accurate sensitivity analysis without resorting to lattice-structure approximations [136, 135] or nested simulations, which could lead to another direction of extension for PEMC.

Finally, although we have focused on standard Monte Carlo sampling, combining PEMC with Quasi Monte Carlo (QMC) techniques may offer further variance reduction benefits [33]. However, it is important to note that QMC methods introduce a low-discrepancy bias, thus forgoing the unbiasedness property of pure Monte Carlo. Balancing unbiasedness with the additional variance reduction from QMC is an intriguing direction for future research.

5.3.3. *Ambulance Diversion Policies Evaluation.* During the early COVID-19 surge, New York City's EMS system faced a drastic spatial shift in emergency calls, pushing certain hospitals to crisis-level overload [42, 46]. In response, a load-balancing rule was devised to divert non-critical ambulance patients away from the closest hospital if that facility nears capacity, balancing travel time against queue length through an optimization-based assignment approach [46]. A stochastic simulation based on historical EMS data [42] was used to evaluate how load-balancing reduces peak occupancy and hospital congestion—showcasing how simulation can improve patient outcomes in EMS networks.

The simulation workflow spans nested simulation system. We focus here on a representative subproblem—the two-hospital ambulance diversion— to illustrate PEMC's advantage. The discrete-event simulation models two emergency departments, ED-1 and ED-2, each with physicians, patients, and arrival streams whose rates $\lambda_{h,d}$ vary by hour h and day d. Ingredients include nonhomogeneous Poisson process with hourly arrival/service rates $\lambda_{h,d}$, e.g., $\text{Arrivals}(t) \sim \text{NHPP}\big(\lambda(t)\big)$; Triage levels and service times where each patient is assigned a triage level $\ell \in \{1, 2, 3, 4, 5\}$ via a (possibly crisis-adjusted) multinomial distribution; Priority queueing and threshold diversion where physicians are modeled as resources and patients forms priority queue ordered by triage. A threshold τ controls ambulance diversion: if the queue Q in one hospital exceeds τ, new ambulance arrivals divert to the other hospital, incurring additional travel time Δ and so on. The evaluation needed is how weekly mortality counts depends on the diversion threshold τ. By varying τ in the simulation, we obtain mortality counts (or rates) as a function of policy choices, enabling downstream decisions. For PEMC predictor we employ a random forest estimator—well-suited for context [35, 108]. We gather a training size 10^5, and sample $\boldsymbol{\theta}$ from a $\boldsymbol{\Theta}$ calibrated from EMS data. The $\boldsymbol{X}$ are readily simulated components for the nested MC—drawing on Weibull, Exponential, Gamma, and Poisson distributions with $\dim(\boldsymbol{X}) = 12$ -and is also parallelizable. The results are summarized Figure 5 and Table7. Further details are left in Appendix C.

FIGURE 5. Ambulance diversion policies evaluation

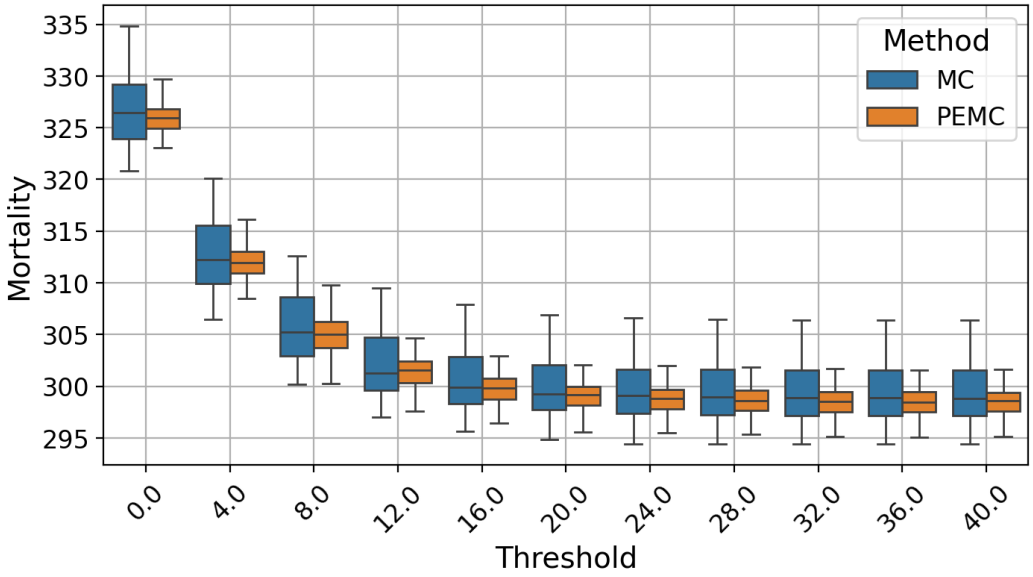

TABLE 7. MSE and MAE for mortality evaluation at various τ.

Threshold	**MSE** $\downarrow$		**MAE** $\downarrow$	
	MC	PEMC	MC	PEMC
$n = 0$	10.974	4.142	2.765	1.587
$n = 20$	9.535	3.266	2.497	1.416
$n = 40$	9.174	3.122	2.428	1.376

6. CONCLUSION

In this paper, we introduced Prediction-Enhanced Monte Carlo (PEMC), a general framework that uses machine learning–based predictors as flexible control variates. By integrating inexpensive, highly parallelizable simulations as features, PEMC preserves the unbiasedness and error quantification of classical Monte Carlo while delivering substantial variance and runtime reductions in settings where traditional variance reduction techniques are infeasible. Looking ahead, several promising research directions arise. First, meta-learning approaches could enable PEMC to automatically learn effective control variates across related tasks, reducing manual tuning burdens [54]. Second, recent advances in generative modeling—such as consistency models and diffusion-based methods [71, 123]—suggest the possibility of generating advanced proposal distributions or learned variates, though integration into finance requires new theoretical developments. Third, robust regression techniques from causal inference and distribution-shift literature could enhance PEMC's stability under model misspecification or nonstationary data [88, 20, 49, 78, 69, 6, 112, 17, 126]. Incorporating ideas from doubly robust estimators and orthogonal statistical learning may yield hybrid methods that further safeguard against bias. Finally, PEMC's versatility invites applications beyond standard derivatives pricing: accelerating robust valuation of exotic options [101, 127], exploring equilibrium asset pricing with transaction costs [96], and solving mean field game equilibria [100]. More ambitiously, PEMC could tackle complex financial control problems—such as optimal control of debt management [25, 26], Stackelberg equilibria in stochastic games [28, 27], and bank salvage modeling via impulse controls [38].

Acknowledgement: We would like to thank [acknowledge individuals, organizations, or institutions] for their valuable input and support throughout this research.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems, 2016. arXiv preprint arXiv:1603.04467.

[2] Christoph Aistleitner, Jozsef Beck, Dmitriy Bilyk, Josef Dick, Michael Drmota, Henri Faure, Peter Hellekalek, Gerhard Larcher, Gunther Leobacher, Dirk Nuyens, et al. *Uniform distribution and Quasi-Monte Carlo methods: discrepancy, integration and applications*, volume 15. Walter de Gruyter GmbH & Co KG, 2014.

[3] Leif Andersen. Simple and efficient simulation of the heston stochastic volatility model. *Journal of Computational Finance*, 11(3):1–43, 2008.

[4] Leif Andersen, Darrell Duffie, and Yang Song. Xva challenges. *Quantitative Finance*, 19(1):1–38, 2017.

[5] Leif BG Andersen and Vladimir V Piterbarg. Interest rate modeling. *Atlantic Financial Press*, 1, 2010.

[6] Anastasios N Angelopoulos, Stephen Bates, Clara Fannjiang, Michael I Jordan, and Tijana Zrnic. Prediction-powered inference. *Science*, 382(6671):669–674, 2023.

[7] Martin Anthony and Peter L. Bartlett. *Neural Network Learning: Theoretical Foundations*. Cambridge University Press, 1999.

[8] Sanjeev Arora, Rong Ge, Tengyu Ma, and Yi Zhang. Stronger generalization bounds for deep nets via a compression approach. In *International Conference on Machine Learning (ICML)*, pages 254–263, 2018.

[9] Søren Asmussen and Peter W Glynn. *Stochastic Simulation: Algorithms and Analysis*. Springer, New York, 2007.

[10] Heejung Bang and James M. Robins. Doubly robust estimation in missing data and causal inference models. *Biometrics*, 61(4):962–973, 2005.

[11] Peter L. Bartlett, Dylan J. Foster, and Matus Telgarsky. Spectrally-normalized margin bounds for neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 6240–6249, 2017.

[12] Peter L Bartlett and Shahar Mendelson. Rademacher and gaussian complexities: Risk bounds and structural results. *Journal of Machine Learning Research*, 3(Nov):463–482, 2002.

[13] Peter L. Bartlett and Shahar Mendelson. Local rademacher complexities. *Annals of Statistics*, 33(4):1497–1537, 2005.

[14] Christian Bayer, Emanuel Derman, Iain Gourlay, and Michael Scott. Variance swaps and options on variance. *Risk Magazine*, 12(9):75–78, 1999.

[15] Christian Bayer and Benoît Stemper. Deep pricing: Financial derivative pricing with deep learning. *Journal of Mathematics in Industry*, 9(1):1–15, 2019.

[16] Aharon Ben-Tal, Laurent El Ghaoui, and Arkadi Nemirovski. Robust optimization: methodology and applications. *Mathematical Programming*, 114(1):1–33, 2008.

[17] Steffen Bickel, Michael Brückner, and Tobias Scheffer. Discriminative learning under covariate shift. *Journal of Machine Learning Research*, 10:2137–2155, 2009.

[18] Jose Blanchet, Haoxuan Chen, Yiping Lu, and Lexing Ying. When can regression-adjusted control variate help? rare events, sobolev embedding and minimax optimality. *Advances in Neural Information Processing Systems*, 36, 2024.

[19] Jose H Blanchet, Peter W Glynn, and Yanan Pei. Unbiased multilevel monte carlo: Stochastic optimization, steady-state simulation, quantiles, and other applications. *arXiv preprint arXiv:1904.09929*, 2019.

[20] Lisa M. Bodnar, Amy R. Cartus, Emily H. Kennedy, Sharon I. Kirkpatrick, Sara M. Parisi, Katherine P. Himes, Courtney B. Parker, William A. Grobman, Hyagriv N. Simhan, Robert M. Silver, et al. Use of a doubly robust machine-learning–based approach to evaluate body mass index as a modifier of the association between fruit and vegetable intake and preeclampsia. *American Journal of Epidemiology*, 191(8):1396–1406, 2022.

[21] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[22] Pierre Boyeau, Anastasios N Angelopoulos, Nir Yosef, Jitendra Malik, and Michael I Jordan. Autoeval done right: Using synthetic data for model evaluation. *arXiv preprint arXiv:2403.07008*, 2024.

[23] Phelim Boyle, Mark Broadie, and Paul Glasserman. Monte carlo methods for security pricing. *Journal of economic dynamics and control*, 21(8-9):1267–1321, 1997.

[24] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, Skye Wanderman-Milne, and Qiao Zhang. JAX: Composable transformations of Python+NumPy programs. https://github.com/google/jax, 2018.

[25] Alberto Bressan and Yilun Jiang. Optimal open-loop strategies in a debt management problem. *Analysis and Applications*, 16(01):133–157, 2018.

[26] Alberto Bressan and Yilun Jiang. The vanishing viscosity limit for a system of hj equations related to a debt management problem. *Discrete & Continuous Dynamical Systems-Series S*, 11(5), 2018.

[27] Alberto Bressan and Yilun Jiang. On the generic structure and stability of stackelberg equilibria. *Journal of Optimization Theory and Applications*, 183:840–880, 2019.

[28] Alberto Bressan and Yilun Jiang. Self-consistent feedback stackelberg equilibria for infinite horizon stochastic games. *Dynamic Games and Applications*, 10(2):328–360, 2020.

[29] Damiano Brigo and Fabio Mercurio. *Interest rate models-theory and practice: with smile, inflation and credit*. Springer Science & Business Media, 2006.

[30] Mark Broadie and Paul Glasserman. Estimating security price derivatives using simulation. *Management Science*, 42(2):269–285, 1996.

[31] Mark Broadie and Özgür Kaya. Exact simulation of stochastic volatility and other affine jump diffusion processes. *Operations research*, 54(2):217–231, 2006.

[32] Hans Buehler, Lukas Gonon, Josef Teichmann, and Ben Wood. Deep hedging. *Quantitative Finance*, 19(8):1271–1291, 2019.

[33] Russel E Caflisch. Monte carlo and quasi-monte carlo methods. *Acta numerica*, 7:1–49, 1998.

[34] René A Carmona. Hjm: A unified approach to dynamic models for fixed income, credit and equity markets. *Paris-Princeton Lectures on Mathematical Finance 2004*, pages 1–50, 2007.

[35] Victor Chang and Timothy Chew. Random forest triage for pre-hospital ambulance data analysis. In *Proceedings of the IEEE International Congress on Big Data (BigData Congress)*, pages 1–6, 2016.

[36] Victor Chernozhukov, Denis Chetverikov, Mert Demirer, Esther Duflo, Christian Hansen, Whitney Newey, and James Robins. Double/debiased machine learning for treatment and structural parameters. *The Econometrics Journal*, 21(1):C1–C68, 2018.

[37] Thomas F Coleman, Yuying Li, and Arun Verma. Interpolation of implied volatility surfaces. *Computational Economics*, 17:203–222, 2001.

[38] Francesco Giuseppe Cordoni, Luca Di Persio, and Yilun Jiang. A bank salvage model by impulse stochastic controls. *Risks*, 8(2):60, 2020.

[39] John C Cox, Jonathan E Ingersoll Jr, and Stephen A Ross. A theory of the term structure of interest rates. *Econometrica*, pages 385–407, 1985.

[40] Jaksa Cvitanic and Fernando Zapatero. *Introduction to the Economics and Mathematics of Financial Markets*. MIT Press, Cambridge, MA, 2004.

[41] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989.
[42] Enrique Lelo de Larrea, Henry Lam, Elioth Sanabria, Jay Sethuraman, Sevin Mohammadi, Audrey Olivier, Andrew W. Smyth, Edward M. Dolan, Nicholas E. Johnson, Timothy R. Kepler, Afsan Quayyum, and Kathleen S. Thomson. Simulating new york city hospital load balancing during covid-19. In *2021 Winter Simulation Conference (WSC)*, pages 1–12, 2021.
[43] Emanuel Derman and Amir E. Fantazzini. Pricing and hedging variance swaps. *Risk Magazine*, 12(8):100–107, 1999.
[44] Emanuel Derman and Iraj Kani. Riding on a smile. *Risk*, 7(2):32–39, 1994.
[45] Kemal Dincer Dingec, Halis Sak, and Wolfgang Hörmann. Variance reduction for asian options under a general model framework. *Review of Finance*, 19(2):907–949, 2015.
[46] Edward Dolan, Nicholas Johnson, Timothy Kepler, Henry Lam, Enrique Lelo de Larrea, Sevin Mohammadi, Audrey Olivier, Afsan Quayyum, Elioth Sanabria, Jay Sethuraman, et al. Hospital load balancing: A data-driven approach to optimize ambulance transports during the covid-19 pandemic in new york city. *Available at SSRN 4094485*, 2022.
[47] Bruno Dupire et al. Pricing with a smile. *Risk*, 7(1):18–20, 1994.
[48] Gintare Karolina Dziugaite and Daniel M. Roy. Computing nonvacuous generalization bounds for deep (and shallow) learning with a pac-bayesian approach. In *Uncertainty in Artificial Intelligence (UAI)*, 2017.
[49] Paul B. Ellickson, Wenyu Kar, and John C. Reeder III. Estimating marketing component effects: Double machine learning from targeted digital promotions. *Marketing Science*, 42(4):704–728, 2023.
[50] Markus Emsermann and Burton Simon. Improving simulation efficiency with quasi control variates. *Stochastic Models*, 18(3):425–448, 2002.
[51] Peyman Mohajerin Esfahani and Daniel Kuhn. Data-driven distributionally robust optimization using the wasserstein metric: Performance guarantees and tractable reformulations. *Mathematical Programming*, 171(1):115–166, 2018.
[52] Benjamin Eyre and David Madras. Auto-evaluation with few labels through post-hoc regression. *arXiv preprint arXiv:2411.12665*, 2024.
[53] Myles Ferguson and Alexander Liang. Deep learning for pricing and hedging American-style options. *arXiv preprint arXiv:1812.11033*, 2018.
[54] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning (ICML)*, pages 1126–1135, 2017.
[55] Dylan J Foster and Vasilis Syrgkanis. Orthogonal statistical learning. *The Annals of Statistics*, 51(3):879–908, 2023.
[56] Michele J. Funk, Daniel Westreich, Christopher Wiesen, Til Stürmer, Alan M. Brookhart, and Marie Davidian. Doubly robust estimation of causal effects. *American Journal of Epidemiology*, 173(7):761–767, 2011.
[57] Jim Gatheral. *The Volatility Surface: A Practitioner's Guide*. John Wiley & Sons, 2006.
[58] Michael B Giles. Multilevel monte carlo path simulation. *Operations research*, 56(3):607–617, 2008.
[59] Michael B. Giles. Multilevel monte carlo methods. *Acta Numerica*, 24:259–328, 2015.
[60] Mike Giles and Paul Glasserman. Smoking adjoints: Fast monte carlo greeks. *Risk*, 19(1):88–92, 2006.
[61] Paul Glasserman. *Monte Carlo methods in financial engineering*. Springer Science & Business Media, 2013.
[62] Paul Glasserman. *Monte Carlo Methods in Financial Engineering*, volume 53 of *Stochastic Modelling and Applied Probability*. Springer, 2013.

[63] Andrew Green, Chris Kenyon, and Chris Dennis. Xva: Credit, funding and capital valuation adjustments. *John Wiley & Sons*, 2015.

[64] Julien Guyon and Pierre Henry-Labordère. *Nonlinear Option Pricing*. Chapman and Hall/CRC, 2014.

[65] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778. IEEE, 2016.

[66] David Heath, Robert Jarrow, and Andrew Morton. Bond pricing and the term structure of interest rates: A new methodology for contingent claims valuation. *Econometrica*, pages 77–105, 1992.

[67] Shane G. Henderson and Peter W. Glynn. Optimization of control variate estimators. *Operations Research*, 50(2):362–371, 2002.

[68] Shane G. Henderson and Barry Simon. Adaptive control variates for monte carlo simulation. *Naval Research Logistics*, 51(4):348–364, 2004.

[69] Miguel A. Hernán and James M. Robins. *Causal Inference: What If*. Chapman & Hall/CRC, Boca Raton, 2020.

[70] Steven L Heston. A closed-form solution for options with stochastic volatility with applications to bond and currency options. *The review of financial studies*, 6(2):327–343, 1993.

[71] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 6840–6851, 2020.

[72] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991.

[73] Béla Horváth, Aitor Muguruza, and Mehdi Tomas. Deep learning volatility: a deep neural network perspective on pricing and calibration in (rough) volatility models. *Quantitative Finance*, 21(1):11–27, 2021.

[74] John C Hull. *Options, Futures, and Other Derivatives*. Pearson, Boston, 2018.

[75] James M Hutchinson, Andrew W Lo, and Tomaso Poggio. A nonparametric approach to pricing and hedging derivative securities via learning networks. *The Journal of Finance*, 49(3):851–889, 1994.

[76] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456. PMLR, 2015.

[77] Faisal Kamiran and Toon Calders. Data preprocessing techniques for classification without discrimination. *Knowledge and Information Systems*, 33(1):1–33, 2012.

[78] Joseph D. Y. Kang and Joseph L. Schafer. Demystifying double robustness: A comparison of alternative strategies for estimating a population mean from incomplete data. *Statistical Science*, 22(4):523–539, 2007.

[79] Ioannis Karatzas and Steven E. Shreve. *Brownian Motion and Stochastic Calculus*, volume 113 of *Graduate Texts in Mathematics*. Springer Science & Business Media, 2nd edition, 1991.

[80] Samuel H. Kim and Shane G. Henderson. Adaptive control variates for pricing multi-asset options. *Journal of Computational Finance*, 10(2):57–82, 2007.

[81] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[82] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, 25:1097–1105, 2012.

[83] Henry Lam and Haofeng Zhang. Doubly robust stein-kernelized monte carlo estimator: Simultaneous bias-variance reduction and supercanonical convergence. *Journal of Machine Learning Research*, 24(85):1–58, 2023.

[84] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[85] Pierre L'Ecuyer and Victor Luu. Neural network-based control variates for option pricing. In *2019 Winter Simulation Conference (WSC)*, pages 3318–3329. IEEE, 2019.

[86] Guillaume Leluc, Maxime Rossi, and David Bolin. Learning control variates for monte carlo methods. *arXiv preprint arXiv:2112.01003*, 2021.

[87] Moshe Leshno, Vedat Y. Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6(6):861–867, 1993.

[88] Fengpei Li and Henry Lam. Robust covariate shift regression. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[89] Z. Li and Y. Zhang. Stein control variates for monte carlo. *Journal of Computational Physics*, 476:111630, 2023.

[90] Han Lin, Haoxian Chen, Krzysztof M Choromanski, Tianyi Zhang, and Clement Laroche. Demystifying orthogonal monte carlo and beyond. *Advances in Neural Information Processing Systems*, 33:8030–8041, 2020.

[91] Sifan Liu. Langevin quasi-monte carlo. *Advances in Neural Information Processing Systems*, 36:75338–75353, 2023.

[92] Roger Lord, Remmert Koekkoek, and Dick Van Dijk. A comparison of biased simulation schemes for stochastic volatility models. *Quantitative Finance*, 10(2):177–194, 2010.

[93] Zhiyuan Lu, Huan Pu, Fei Wang, Zhiqiang Hu, and Li Wang. The expressive power of neural networks: A view from the approximation theory. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 440–448, 2017.

[94] Stéphane Maire. Monte carlo integration by l^2-function approximation. *Journal of Computational and Applied Mathematics*, 151:187–199, 2003.

[95] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. MIT Press, 2nd edition, 2018.

[96] Johannes Muhle-Karbe, Marcel Nutz, and Xiaowei Tan. Asset pricing with heterogeneous beliefs and illiquidity. *Mathematical Finance*, 30(4):1392–1421, 2020.

[97] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010.

[98] Barry L Nelson. Control variate remedies. *Operations Research*, 38(6):974–992, 1990.

[99] Behnam Neyshabur, Zhiyuan Li, Srinadh Bhojanapalli, Yann LeCun, and Nathan Srebro. A pac-bayesian approach to spectrally-normalized margin bounds for neural networks. In *International Conference on Learning Representations (ICLR)*, 2018.

[100] Marcel Nutz, Jaime San Martin, and Xiaowei Tan. Convergence to the mean field game limit: a case study. *The Annals of Applied Probability*, 30(1):259–286, 2020.

[101] Marcel Nutz, Florian Stebegg, and Xiaowei Tan. Multiperiod martingale transport. *Stochastic Processes and their Applications*, 130(3):1568–1615, 2020.

[102] Chris J. Oates, Mark Girolami, and Nicolas Chopin. Control functionals for monte carlo integration. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 79(3):695–718, 2017.

[103] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[104] Raghu Pasupathy, Bruce W Schmeiser, Michael R Taaffe, and Jin Wang. Control-variate estimation using estimated control means. *Iie Transactions*, 44(5):381–385, 2012.

[105] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative

style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32, pages 8024–8035. Curran Associates, Inc., 2019.

[106] Benjamin Peherstorfer, Karen Willcox, and Max Gunzburger. Survey of multifidelity methods in uncertainty propagation, inference, and optimization. *Siam Review*, 60(3):550–591, 2018.

[107] Allan Pinkus. Approximation theory of the mlp model in neural networks. *Acta Numerica*, 8:143–195, 1999.

[108] Romain Pirracchio, Matthieu Resche-Rigon, and Sylvie Chevret. Mortality prediction in intensive care units with the random forest model. *BMC Medical Research Methodology*, 15(1):1–10, 2015.

[109] François Portier and Johan Segers. Monte carlo integration with a growing number of control variates. *Journal of Applied Probability*, 55(4):1078–1092, 2018.

[110] Thomas P Prescott and Ruth E Baker. Multifidelity approximate bayesian computation. *SIAM/ASA Journal on Uncertainty Quantification*, 8(1):114–138, 2020.

[111] Philip Protter. *Variance and Volatility Swaps*. Wiley Finance, Hoboken, NJ, 2010.

[112] Joaquin Quionero-Candela, Masashi Sugiyama, Anton Schwaighofer, and Neil D. Lawrence. Dataset shift in machine learning. In *Dataset Shift in Machine Learning*. The MIT Press, Cambridge, MA, 2009.

[113] Alok Rajkomar, Moritz Hardt, and Michael D. Howell. Ensuring fairness in machine learning to advance health equity. *Annals of Internal Medicine*, 169(12):866–872, 2018.

[114] Yong Ren, Dilip Madan, and M Qian Qian. Calibrating and pricing with embedded local volatility models. *RISK-LONDON-RISK MAGAZINE LIMITED-*, 20(9):138, 2007.

[115] Chang-han Rhee and Peter W Glynn. Unbiased estimation with square root convergence for sde models. *Operations Research*, 63(5):1026–1043, 2015.

[116] James M. Robins, Andrea Rotnitzky, and Lue Ping Zhao. Estimation of regression coefficients when some regressors are not always observed. *Journal of the American Statistical Association*, 89(427):846–866, 1994.

[117] Johannes Ruf and Weiguan Wang. Neural networks for option pricing and hedging: A literature review. *The Journal of Computational Finance*, 25(2):1–46, 2021.

[118] Tomasz Rychlik. Unbiased nonparametric estimation of the derivative of the mean. *Statistics & probability letters*, 10(4):329–333, 1990.

[119] Marc Sabate Vidales, David Šiška, and Lukasz Szpruch. Unbiased deep solvers for linear parametric pdes. *Applied Mathematical Finance*, 28(4):299–329, 2021.

[120] Zuowei Shen, Haizhao Yang, and Shijun Zhang. Neural network approximation: Three perspectives. *Foundations of Computational Mathematics*, 21(1):3–59, 2021.

[121] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[122] Justin Sirignano and Konstantinos Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375:1339–1364, 2018.

[123] Yang Song, Jascha Sohl-Dickstein, Durk P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations (ICLR)*, 2021.

[124] Leah South, Simon Barthelmé, and Iain Murray. Regularised control variates for variance reduction. *arXiv preprint arXiv:2202.12023*, 2022.

[125] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[126] Masashi Sugiyama and Motoaki Kawanabe. Covariate shift adaptation by importance weighted cross-validation. In Joaquin Quionero-Candela, Masashi Sugiyama, Anton Schwaighofer, and Neil D. Lawrence, editors, *Dataset Shift in Machine Learning*, pages 123–130. The MIT Press, Cambridge, MA, 2009.

[127] Xiaowei Tan. *Optimal Transport and Equilibrium Problems in Mathematical Finance*. Columbia University, 2019.

[128] Grigore Tataru and Travis Fisher. Stochastic local volatility. *Quantitative Development Group, Bloomberg Version*, 1, 2010.

[129] Vladimir N. Vapnik. *Statistical Learning Theory*. Wiley, 1998.

[130] Oldrich Vasicek. An equilibrium characterization of the term structure. *Journal of Financial Economics*, 5(2):177–188, 1977.

[131] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, pages 5998–6008. Curran Associates, Inc., 2017.

[132] Ruosi Wan, Mingjun Zhong, Haoyi Xiong, and Zhanxing Zhu. Neural control variates for monte carlo variance reduction. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2019, Würzburg, Germany, September 16–20, 2019, Proceedings, Part II*, pages 533–547. Springer, 2020.

[133] Julian Wrede, Helge Wrede, and Wilhelm Behringer. Emergency department mean physician time per patient and workload predictors ED-MPTPP. *J Clin Med*, 9(11), nov 2020.

[134] Dmitry Yarotsky. Error bounds for approximations with deep relu networks. *Neural Networks*, 94:103–114, 2017.

[135] Honglei Zhao, Rupak Chatterjee, Thomas Lonon, and Ionuţ Florescu. Pricing bermudan variance swaptions using multinomial trees. *The Journal of Derivatives*, 26(3):22–34, 2019.

[136] Honglei Zhao, Zhe Zhao, Rupak Chatterjee, Thomas Lonon, and Ionuţ Florescu. Pricing variance, gamma and corridor swaps using multinomial trees. *The Journal of Derivatives*, 25(2):7–21, 2017.

[137] Wendong Zheng and Yue Kuen Kwok. Closed form pricing formulas for discretely sampled generalized variance swaps. *Mathematical Finance*, 24(4):855–881, 2014.

[138] Tijana Zrnic and Emmanuel J Candès. Cross-prediction-powered inference. *Proceedings of the National Academy of Sciences*, 121(15):e2322083121, 2024.

Appendix A. Proofs

A.1. **Proof of Lemma 3.**

Proof. Proof By relaxing the constraints to $n, N \in \mathbb{R}^+$, the optimization problem

$$\begin{aligned} \min_{n>0,N>0} \quad & \frac{1}{n}\sigma_{f-g}^2 + \frac{1}{N}\sigma_g^2. \\ \text{s.t.} \quad & nc_{f-g} + Nc_g \le B, \end{aligned}$$

is jointly convex in $n, N \in \mathbb{R}^+$. The objective value also approaches infinity as $N \to 0$ or $n \to 0$. Consequently, the strict convexity ensures the existence of a global minimizer in the interior and directly solving the Lagrangian gives us the result. □

A.2. **Proof of Lemma 4.**

Proof. Proof Based on the results of Lemma 3, we can deduce that this ratio again does not depend on the budget B, so we omit its dependence in the statement. Moreover, we can conveniently choose a budget of the form $B = n_0 c_{f-g} + n_0 \frac{\sigma_g}{\sigma_{f-g}} \cdot \sqrt{\frac{c_{f-g}}{c_g}} c_g$ which gives an allocation of $n = n_0$ and $N = n_0 \frac{\sigma_g}{\sigma_{f-g}} \cdot \sqrt{\frac{c_{f-g}}{c_g}}$ for PEMC in accordance with Lemma 3, while comparing with B/c_f samples for standard MC. The rest follows from Lemma 2 and the assumption $c_f = c_{f-g}$. □

A.3. **Proof of Lemma 5.**

Proof. Proof When $g = \mathbb{E}[f(\boldsymbol{Y}) \,|\, \boldsymbol{X}]$ is the true regressor, i.e.,

$$g = \underset{h \text{ measurable}}{\arg\min} \; \mathbb{E}[(f(\boldsymbol{Y}) - h(\boldsymbol{X}))^2],$$

and f is square-integrable, it follows that $\mathbb{E}[(f-g)h] = 0$ for all square-integrable h. Plugging in $h = g$, we obtain $\mathrm{Cov}(f-g, g) = 0$. Consequently we have $\rho := \mathrm{corr}(f, g) = \frac{\sigma_g}{\sigma_f}$ and $\sigma_f^2 = \sigma_{f-g}^2 + \sigma_g^2$, which further implies $\sigma_g = \rho^2 \sigma_f^2$ and $\sigma_{f-g}^2 = (1-\rho^2)\sigma_f^2$. The rest follows from $c_{f-g} = c_f$. □

A.4. **Proof of Lemma 6.**

In this part, we go through the pipeline for a result in the form of Lemma 6 in Section 4.3.1. We begin by formally introducing the key concepts associated with the learning framework under consideration.

Definition 4. *Let g_0 denote the true regression function, defined by*

$$\begin{aligned} g_0(\boldsymbol{\theta}, \boldsymbol{X}(\boldsymbol{\theta})) :=& \mathbb{E}_{\textit{risk neutral measure}(\boldsymbol{\theta})}\left[f_{\boldsymbol{\theta}_{\textit{payoff}}}(\boldsymbol{Y}(\boldsymbol{\theta})) \,\middle|\, (\boldsymbol{\theta}, \boldsymbol{X}(\boldsymbol{\theta})) \right] \\ =& \underset{g \textit{ measurable}}{\arg\min} \; \mathbb{E}_{\boldsymbol{\theta}\sim\Theta, (\boldsymbol{X}(\boldsymbol{\theta}), \boldsymbol{Y}(\boldsymbol{\theta})) \sim \textit{risk neutral measure}(\boldsymbol{\theta})} \left(f_{\boldsymbol{\theta}_{\textit{payoff}}}(\boldsymbol{Y}(\boldsymbol{\theta})) - g(\boldsymbol{\theta}, \boldsymbol{X}(\boldsymbol{\theta})) \right)^2. \end{aligned}$$

Note that equality exploits the well-known property of the squared loss minimizer: it is the conditional expectation of the target variable given the input features. The notable part is that, this formulation inherently incorporates the sampling over $\boldsymbol{\theta} \sim \boldsymbol{\Theta}$. This is because, in PEMC we generate training data by first drawing $\boldsymbol{\theta}$ from $\boldsymbol{\Theta}$ and then sampling $\boldsymbol{X}(\boldsymbol{\theta}), \boldsymbol{Y}(\boldsymbol{\theta})$ from the corresponding risk-neutral measure indexed by $\boldsymbol{\theta}$. However, since the result holds pointwise for each fixed $\boldsymbol{\theta} \in \boldsymbol{\Theta}$, it also holds in the aggregate setting where $\boldsymbol{\theta}$ is (uniformly) random. For simplicity, we do not delve into technical measurability considerations here.

Next, we introduce the hypothesis class and define the best-in-class predictor. The hypothesis class $\mathcal{G}$ plays a pivotal role in the learning framework, encapsulating the set of candidate functions from which the predictor g is selected.

Definition 5. *Let $\mathcal{G}$ be the hypothesis class induced by the NN model family. Define g^* as the best-in-class function satisfying*

$$g^*(\boldsymbol{\theta}, \boldsymbol{X}(\boldsymbol{\theta})) := \arg\min_{g\in\mathcal{G}} \mathbb{E}_{\boldsymbol{\theta}\sim\Theta,(\boldsymbol{X}(\boldsymbol{\theta}),\boldsymbol{Y}(\boldsymbol{\theta}))\sim \textit{risk neutral measure}(\boldsymbol{\theta})} \left(f_{\boldsymbol{\theta}_{payoff}}(\boldsymbol{Y}(\boldsymbol{\theta})) - g(\boldsymbol{\theta}, \boldsymbol{X}(\boldsymbol{\theta})) \right)^2$$
$$= \arg\min_{g\in\mathcal{G}} \mathbb{E}_{\boldsymbol{\theta}\sim\Theta,\boldsymbol{X}(\boldsymbol{\theta})\sim \textit{risk neutral measure}(\boldsymbol{\theta})} \left((g_0 - g)(\boldsymbol{\theta}, \boldsymbol{X}(\boldsymbol{\theta})) \right)^2.$$

where the second line follows again from the definition of g_0. Considering a model g trained with N_{train} samples and is held fixed during the evaluation of the expectation, as is standard when discussing generalization error, the approximation error is defined as

$$\epsilon_a^{\mathcal{G}} := \mathbb{E}_{\boldsymbol{\theta}\sim\Theta,\boldsymbol{X}(\boldsymbol{\theta})\sim \textit{risk neutral measure}(\boldsymbol{\theta})} \left((g_0 - g^*)(\boldsymbol{\theta}, \boldsymbol{X}(\boldsymbol{\theta})) \right)^2,$$

and the statistical error from g obtained from training on N_{train} samples:

$$\epsilon_e^{N_{train}} := \mathbb{E}_{\boldsymbol{\theta}\sim\Theta,\boldsymbol{X}(\boldsymbol{\theta})\sim \textit{risk neutral measure}(\boldsymbol{\theta})} \left((g^* - g)(\boldsymbol{\theta}, \boldsymbol{X}(\boldsymbol{\theta})) \right)^2.$$

Finally, define the total error as

$$\epsilon_{total} := \mathbb{E}_{\boldsymbol{\theta}\sim\Theta,\boldsymbol{X}(\boldsymbol{\theta})\sim \textit{risk neutral measure}(\boldsymbol{\theta})} \left((g_0 - g)(\boldsymbol{\theta}, \boldsymbol{X}(\boldsymbol{\theta})) \right)^2.$$

It follows that

$$\epsilon_{\text{total}} \leq 2\epsilon_a^{\mathcal{G}} + 2\epsilon_e^{N_{\text{train}}}. \tag{21}$$

The interaction between $\epsilon_a^{\mathcal{G}}$ and $\epsilon_e^{N_{\text{train}}}$ essentially captures the bias-variance tradeoff. By choosing more expressive model classes (e.g., richer neural network architectures) and increasing the training set size, we can jointly reduce $\epsilon_a^{\mathcal{G}}$ and $\epsilon_e^{N_{\text{train}}}$. Thus, under reasonable conditions, ϵ_{total} can be made arbitrarily small if both $\epsilon_a^{\mathcal{G}}$ and $\epsilon_e^{N_{\text{train}}}$. In the following sections, we analyze $\epsilon_a^{\mathcal{G}}$ and $\epsilon_e^{N_{\text{train}}}$ respectively.

A.4.1. *Approximation Error $\epsilon_a^{\mathcal{G}}$.* Note that this best-in-class function g^* depends on the distribution of $\boldsymbol{\theta} \sim \Theta$, meaning the notion of optimality is distribution-dependent on $\boldsymbol{\theta}$. This is also why we choose a distribution $\boldsymbol{\theta} \sim \boldsymbol{\Theta}$ that sufficiently covers the space. In this paper, $\mathcal{G}$ is induced by our choice of neural network architecture and training procedure. The complexity of $\mathcal{G}$ significantly impacts both approximation and statistical errors. Neural networks are renowned for their flexibility and expressiveness, serving as universal approximators. According to the Universal Approximation Theorem [41, 72], neural networks with a single hidden layer containing a sufficient number of neurons can approximate any continuous function on compact subsets of $\mathbb{R}^n$ to arbitrary accuracy. Modern extensions of this theorem provide more nuanced insights into how network depth and architecture influence approximation capabilities [134, 93, 120, 41, 72]. Typical universal universal approximation theorems deals with compact input space in $\mathbb{R}^n$ and point-wise convergence, in our context, we do not restrict $\boldsymbol{X}$ to live in a compact space and we only need $\mathscr{L}^2$ convergence, so the theorem from [107] or [87] is sufficient, which gives us

Lemma 7. *Suppose $g_0 : \boldsymbol{\Theta} \times \mathbb{R}^{dim(\boldsymbol{X})} \to \mathbb{R}$ is in $\mathscr{L}^2(\mathbb{P})$ where $\mathbb{P}$ is the probability measure that governs the joint distribution of $\boldsymbol{\theta} \sim \Theta, (\boldsymbol{X}(\boldsymbol{\theta}), \boldsymbol{Y}(\boldsymbol{\theta})) \sim$ risk neutral measure$(\boldsymbol{\theta})$. Then, for any $\epsilon > 0$, we can find a class of NN $\mathcal{G}$ such that*

$$\epsilon_a^{\mathcal{G}} \leq \epsilon.$$

Proof. Proof See [107] or [87]. □

A.4.2. *Statistical Error* $\epsilon_e^{N_{train}}$. To quantify the capacity of $\mathcal{G}$, one could employ common complexity measures such as the Vapnik-Chervonenkis (VC) dimension [129] or Rademacher complexity [12]. These measures provide bounds on the generalization error by capturing the richness of the hypothesis class. Modern studies have further refined our understanding of neural network complexity. For instance, norm-based capacity controls [11], PAC-Bayesian bounds [48], and local Rademacher complexities [13] offer more nuanced insights. In particular, most neural network architectures are known to have finite complexity measures, such as finite VC dimension or other capacity metrics, when considered as a hypothesis class with a fixed number of parameters [7].

Consequently, when empirical risk minimization (ERM) is performed over these networks with controlled complexity, standard statistical learning theory guarantees apply, yielding generalization bounds that typically decrease on the order of $O(1/\sqrt{N_{\text{train}}})$ where N_{train} is the sample size [11, 99, 8] and the constant in the rate depends on the complexity measure. Our PEMC prediction model can be considered as a ERM estimator. Thus, these results ensure that, given sufficiently large training sets and appropriate capacity constraints (e.g., weight regularization or architectural choices), neural networks can achieve low statistical error $\epsilon_e^{N_{\text{train}}}$.

Finally, there is an optimization error arising from the discrepancy between the empirical risk minimizer within $\mathcal{G}$ and the final trained model g. This error accounts for situations where the optimization algorithm does not perfectly identify the empirical risk minimizer. However, we do not consider this optimization error, effectively assuming the presence of an ideal "oracle" for optimization. Consequently, theoretical analyses typically treat our g as readily available empirical minimizer.

Lemma 8. *Suppose g is the empirical risk minimizer of Algorithm 1 and the neural network class $\mathcal{G}$ has a finite VC-dimension or Rademacher complexity. Then, for any $\epsilon > 0$, there exists N_{train} such that*

$$\epsilon_e^{N_{train}} \leq \epsilon.$$

A.4.3. *Proof of Lemma 6.* Lemma 7 and Lemma 8 allows us to control ϵ_{total} using (21), which allows us to prove Lemma 6 with the help of the following technical definition and lemma.

Definition 6. *Define* $\epsilon_{total}(\boldsymbol{\theta}) := \mathbb{E}_{\boldsymbol{X}(\boldsymbol{\theta})\sim risk\ neutral\ measure(\boldsymbol{\theta})}\big(g(\boldsymbol{X}(\boldsymbol{\theta})) - g_0(\boldsymbol{X}(\boldsymbol{\theta}))\big)^2$.

As a result, we have $\mathbb{E}_{\boldsymbol{\theta}\sim\boldsymbol{\Theta}}\epsilon_{\text{total}}(\boldsymbol{\theta}) = \epsilon_{\text{total}}$, which allows us to establish the following lemma.

Lemma 9. *Let G be a second moment bound for g, i.e.,*

$$\mathbb{E}_{\boldsymbol{X}(\boldsymbol{\theta})\sim risk\ neutral\ measure(\boldsymbol{\theta})} g^2(\boldsymbol{X}(\boldsymbol{\theta})) \leq G.$$

uniformly for all $\boldsymbol{\theta} \in \boldsymbol{\Theta}$. Then, for each $\boldsymbol{\theta} \in \boldsymbol{\Theta}$,

$$|\sigma_f^2 - \sigma_g^2 - \sigma_{f-g}^2| \leq 2\sqrt{\epsilon_{total}(\boldsymbol{\theta})}\sqrt{G}.$$

Proof. Proof Fix any $\boldsymbol{\theta} \in \boldsymbol{\Theta}$. From the variance decomposition, we have

$$\text{Var}(f) = \text{Var}(g) + \text{Var}(f-g) + 2\,\text{Cov}(f-g, g),$$

which implies

$$|\sigma_f^2 - \sigma_g^2 - \sigma_{f-g}^2| \leq 2|\text{Cov}(f-g, g)|.$$

Conditioning on $\boldsymbol{X}$ and utilizing the properties of g_0, we find that $\text{Cov}(f-g, g) = \text{Cov}(g_0 - g, g)$. Applying the Cauchy–Schwarz inequality yields the desired inequality. □

We can now prove Lemma 6.

Proof. Proof of Lemma 6 Besides the assumptions in Lemma 7, Lemma 8 and Lemma 9, we further assume

$$g \leq \mathbb{E}_{\boldsymbol{X}(\boldsymbol{\theta})\sim\text{risk neutral measure}(\boldsymbol{\theta})} g^2(\boldsymbol{X}(\boldsymbol{\theta})), \mathbb{E}_{\boldsymbol{Y}(\boldsymbol{\theta})\sim\text{risk neutral measure}(\boldsymbol{\theta})} f^2(\boldsymbol{Y}(\boldsymbol{\theta})) \leq G, \tag{22}$$

for some $0 < g < G < \infty$, uniformly for all $\theta \in \Theta$. Then, based on Lemma 7, 8 and Markov inequality, we can find $\mathcal{G}$ and N_{train} such that, with probability at least $1-\delta$, the randomly sampled $\boldsymbol{\theta} \sim \boldsymbol{\Theta}$ satisfies

$$\epsilon_{\text{total}}(\boldsymbol{\theta}) \leq \epsilon,$$

which, together with Lemma 9 gives

$$|\sigma_f^2 - \sigma_g^2 - \sigma_{f-g}^2| \leq O(\epsilon),$$

with the constant in O not dependent on $\boldsymbol{\theta}$. Then, for at least $1-\delta$ fraction of $\boldsymbol{\theta} \in \boldsymbol{\Theta}$, as one shrinks $\epsilon \to 0$, $\epsilon_{\text{total}}(\boldsymbol{\theta}) \leq \epsilon$ and $|\sigma_f^2 - \sigma_g^2 - \sigma_{f-g}^2| \leq O(\epsilon)$ implies $\sigma_g^2 \to \sigma_{g_0}^2$ and $\sigma_{f-g_0}^2 \to \sigma_{f-g}^2$ uniformly for all such $\boldsymbol{\theta}$. Since $\sigma_f^2 = \sigma_{g_0}^2 + \sigma_{f-g_0}^2$, this further implies $\frac{\sigma_g}{\sigma_f} \to \frac{\sigma_{g_0}}{\sigma_f}$ and $\frac{\sigma_{f-g}}{\sigma_f} \to \frac{\sigma_{f-g_0}}{\sigma_f}$, given the boundedness of (22). Thus, $\frac{\text{Var}(PEMC)}{\text{Var}(MC)}$ in (12) $\to r(\rho, c)$ in (13), or equivalently, with probability at least $1-\delta$, the randomly sampled $\boldsymbol{\theta} \sim \boldsymbol{\Theta}$ satisfies

$$\frac{\text{Var}(PEMC)}{\text{Var}(MC)} = r(\rho, c) + O(\epsilon),$$

where the constant in O does not depend on $\boldsymbol{\theta}$. This concludes the proof. □

Appendix B. Numerical Results on Asian Option Pricing

We revisit the Asian option example and conduct experiments using the arithmetic Asian call option under the Geometric Brownian Motion (GBM) model. We choose the GBM model instead of the Heston model (6) to facilitate a direct comparison between PEMC, standard Monte Carlo (MC), and the traditional Control Variate (CV) method on the extent of variance reduction. This is because the geometric Asian option, which serves as a well-established control variate, is analytically tractable within the GBM framework but lacks a closed-form solution under the Heston model [45].

This experiment aims to demonstrate the typical level of variance reduction that PEMC achieves. Our focus is strictly limited to variance reduction, because the GBM model actually allows for highly efficient and parallelized generation of both asset paths and corresponding payoffs, making the simulation of $\boldsymbol{Y}$ computationally straightforward. In other words, one could not fully leverage PEMC's strengths in GBM model and this toy experiment only serves to provide insights into PEMC's effectiveness in a controlled setting where a known CV exists. Practical applications and actual use-cases of PEMC will be provided in Section 5.

B.1. **Experimental Setup.** We implement the PEMC procedure detailed in Section 3.5, utilizing a neural network (NN) as the predictive model. The asset price follows the Geometric Brownian Motion (GBM) model:

$$dS_t = rS_t dt + \sigma S_t dW_t^S. \tag{23}$$

The payoff calculation $P_A(\{S_t\}_t) = (\frac{1}{n_D}\sum_{i=1}^{n_D} S_{t_i} - K, 0)^+$ with $n_D = 252$, where simulations are carried out using daily time increments (i.e., $\Delta t = 1$). The parameter space $\boldsymbol{\theta} := (r, S_0, \sigma, K) \in \boldsymbol{\Theta}$ is: $r \in [0.01, 0.03]$, $S_0 \in [80, 120]$, $\sigma \in [0.05, 0.25]$, and $K \in [90, 110]$. For our feature $\boldsymbol{X}$, we experiment with the whole sum of Brownian increments $W_T^S := \sum_{j=1}^{252} \Delta W_j^S$ with $\dim(\boldsymbol{X}) = 1$. We also tried a more granular representation where we partition 252 Brownian increments evenly into 14 parts, each summing 18 consecutive increments: $[\boldsymbol{X}]_i = \sum_{j=18(i-1)+1}^{18i} \Delta W_j^S$ for $i \in [14]$, yielding $\dim(\boldsymbol{X}) = 14$. The neural network is trained on a dataset comprising $N_{\text{train}} = 1.28 \times 10^6$ samples, generated by uniformly sampling $\boldsymbol{\theta} \sim \boldsymbol{\Theta}$.

The neural network architecture consists of three components: a $\boldsymbol{\theta}$ network branch, a $\boldsymbol{X}$ network branch, and a combined network. This architecture incorporates several modern deep learning practices, including Batch Normalization [76], Rectified Linear Unit (ReLU) activations [97], and skip connections

[65] to enhance training stability and model performance. The $\boldsymbol{\theta}$ network branch processes the 4-dimensional input $\boldsymbol{\theta}$ using two fully connected layers with Batch Normalization and ReLU activation functions, outputting a 10-dimensional feature vector for subsequent processing. The $\boldsymbol{X}$ network branch uses two fully connected layers with width $\max(32, 2\dim(\boldsymbol{X}))$ neurons. Finally, the combined network integrates outputs from both branches through concatenation and employs skip connections featured in ResNet architectures to preserve information flow. The concatenated features pass through two additional fully connected layers with Batch Normalization and ReLU activation to produce the final prediction for PEMC. The network is implemented using PyTorch [105] and trained using the Adam optimizer [81] with a learning rate of 1×10^{-3}, incorporating dropout layers [125] with a rate of 0.5 after each hidden layer to prevent overfitting.

B.2. **Evaluation.** For comparison, we implement PEMC, MC and CV method with geometric Asian option

$$P_G(\{S_t\}_t) = ((\prod_{i=1}^{n_D} S_{t_i})^{\frac{1}{n_D}} - K, 0)^+,$$

for the evaluation at $\boldsymbol{\theta} = (r, S_0, \sigma, K) = (0.02, 100, 0.2, 100)$. The CV estimator is given by

$$\text{CV} = \frac{1}{n}\sum_{i=1}^{n}\Big(P_A(\{S_t^{(i)}\}_t) - P_G(\{S_t^{(i)}\}_t)\Big) + P_G^{\text{exact}}(\boldsymbol{\theta}),$$

where $P_G^{\text{exact}}(\boldsymbol{\theta})$ is the closed-form price of geometric Asian option (with correction regarding n_D) [61]. We first evaluate the mean of 2×10^9 MC samples as the ground truth value A_0. Then, we evaluate n sample average of the MC, PEMC ($N = 10n$) and the geometric CV estimator for $n = 1000, 4000, 9000$ respectively, to record 3 different estimator for A_0. Then we repeat the experiments 300 times to get 300 estimators for each $n \in \{1000, 4000, 9000\}$ and each method in {MC, PEMC, Geometric CV}, to compare against A_0. The performance of the estimators is summarized in Table 8 and Figure 6.

TABLE 8. Root Mean Squared Error from 300 Experiments

Method	$n = 1000$	$n = 4000$	$n = 9000$
Monte Carlo (MC)	0.2376	0.1173	0.0854
PEMC ($\dim \boldsymbol{X} = 1$)	0.1509	0.0809	0.0481
PEMC ($\dim \boldsymbol{X} = 14$)	0.0781	0.0397	0.0261
Geometric CV	0.0099	0.0051	0.0036

Given that all estimators are unbiased, our analysis focuses exclusively on variance-related metrics. The experimental results demonstrate the superior performance of the PEMC framework compared to the standard Monte Carlo (MC) estimator. In particular, in all sample sizes, the PEMC framework delivers visible variance reduction over MC: PEMC's basic variant with $\dim \boldsymbol{X} = 1$ achieves a 30-40% reduction in root mean squared error (RMSE) relative to MC, while the more sophisticated variant with $\dim \boldsymbol{X} = 14$ attains an impressive 65-70% reduction in RMSE over MC. However, the Geometric CV emerges as the most efficient estimator by far, with RMSE an order of magnitude smaller than MC.

To better understand this hierarchy in estimator performance, it is helpful to recognize PEMC not as a specific estimator, but rather as a framework that latches on and enhance an existing baseline. In our current example, we chose standard MC as the baseline, and design our PEMC directly upon MC to improve it. This also illustrates a common state of practice: in most complex scenarios, sophisticated baselines like CV are not available, and practitioners must rely on some variants of naive MC methods. Then, PEMC can be applied to these baselines to achieve variance reduction. However, PEMC is a flexible framework designed to enhance general simulation baselines, including sophisticated ones, not

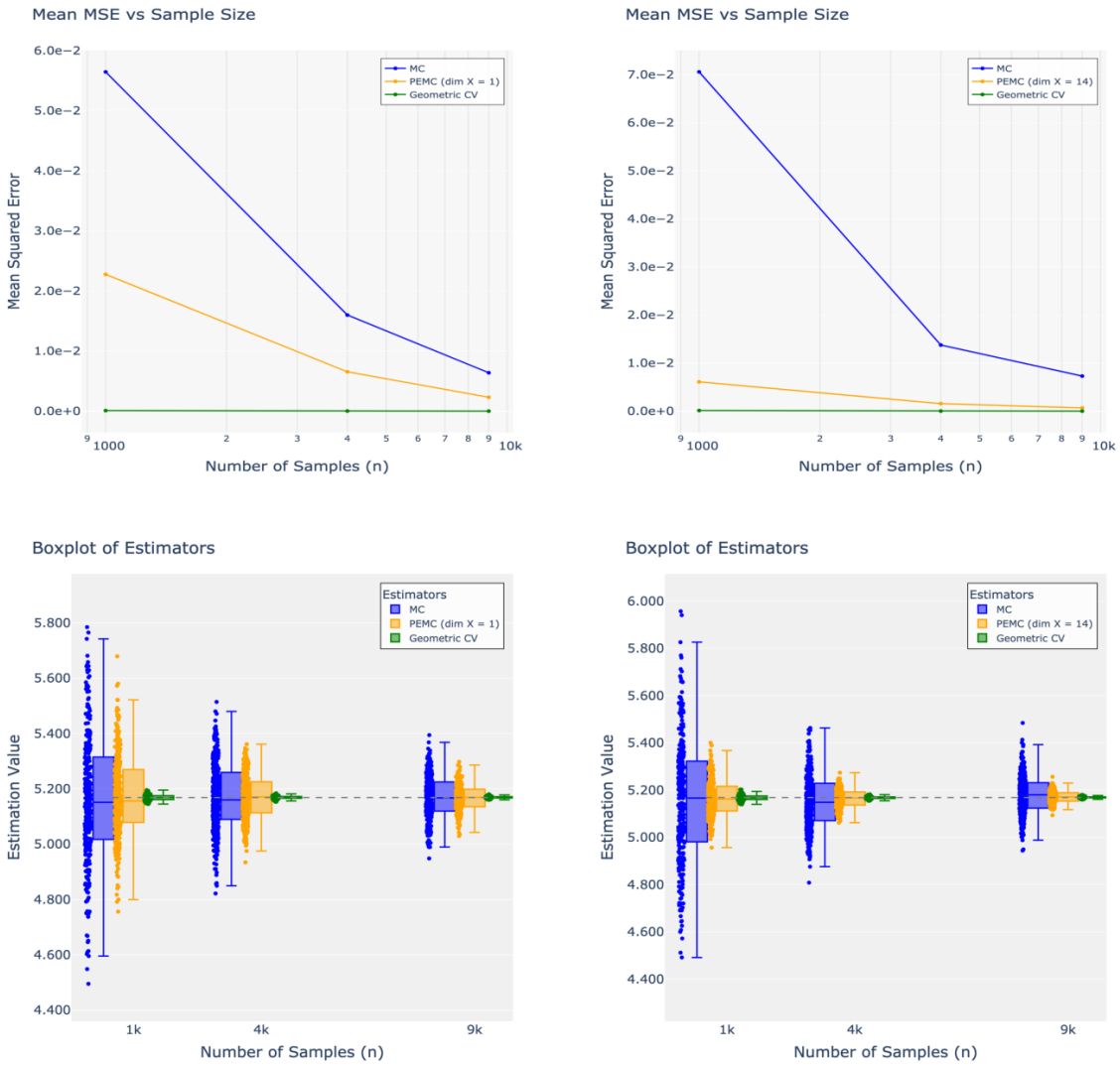

FIGURE 6. Performance of Estimators for Asian Options. The Comparison of estimator across Monte Carlo (MC), PEMC, and Geometric Control Variate (CV) is based on 300 experiments. Top: Mean squared errors (MSE) plot as a function of n for (left) PEMC with $\dim \boldsymbol{X} = 1$ and (right) PEMC with $\dim \boldsymbol{X} = 14$, both compared against MC and Geometric CV. Bottom: Corresponding boxplots of the 300 estimates.

just standard Monte Carlo. In practice, efficient MC techniques such as Quasi-Monte Carlo (QMC) methods [33, 2] and their variants—including Langevin Monte Carlo (LMC) [91], Orthogonal Monte Carlo (OMC), and Near-Orthogonal Monte Carlo (NOMC) [90]—can be integrated with PEMC for further variance reduction.

B.3. **Using PEMC as a "Boost" to Known CV.** Lastly, we demonstrate how PEMC can be used to improve existing CV. This supports the creative use of PEMC for further efficiency enhancement even for problems that already possess some variance reduction approaches. In particular, when a

CV with known mean is available, such as in this current example, PEMC can be built upon it to potentially achieve even greater variance reduction. To see how, in Algorithm 1, we change the label from $P_A(\{S_t\}_t)$ (MC baseline) to $P_A(\{S_t\}_t) - P_G(\{S_t\}_t) + P_G^{\text{exact}}(\boldsymbol{\theta})$ (Geometric CV baseline), and let the model g learn

$$\begin{aligned} g(\boldsymbol{\theta}, \boldsymbol{X}(\boldsymbol{\theta})) \approx & \mathbb{E}_{\text{risk neutral measure}(\boldsymbol{\theta})}\left[P_A(\{S_t\}_t) - P_G(\{S_t\}_t) + P_G^{\text{exact}}(\boldsymbol{\theta}) \middle| (\boldsymbol{\theta}, \boldsymbol{X}(\boldsymbol{\theta}))\right] \\ = & \mathbb{E}_{\text{risk neutral measure}(\boldsymbol{\theta})}\left[P_A(\{S_t\}_t) - P_G(\{S_t\}_t) \middle| (\boldsymbol{\theta}, \boldsymbol{X}(\boldsymbol{\theta}))\right] + P_G^{\text{exact}}(\boldsymbol{\theta}). \end{aligned} \tag{24}$$

This formulation suggests we can train g on the difference of $P_A(\{S_t\}_t) - P_G(\{S_t\}_t)$ as labels and then add back the closed-form $P_G^{\text{exact}}(\boldsymbol{\theta})$ to g. We implemented this approach, which we call "Boost PEMC," using the basic PEMC variant where $\boldsymbol{X} = W_T^S := \sum_{j=1}^{252} \Delta W_j^S$ with $\dim \boldsymbol{X} = 1$. As shown in Table 9 and Figure 7, this Boost PEMC indeed achieves variance reduction over the sophisticated Geometric CV estimator, delivering a 35-40% reduction in RMSE, remarkably consistent with the relative performance gain we observed when applying PEMC to the MC baseline in Figure 6.

TABLE 9. Root Mean Squared Error from 300 Experiments

Method	$n = 1000$	$n = 4000$	$n = 9000$
Boost PEMC ($\dim \boldsymbol{X} = 1$)	0.0065	0.0031	0.0021
Geometric CV	0.0099	0.0051	0.0036

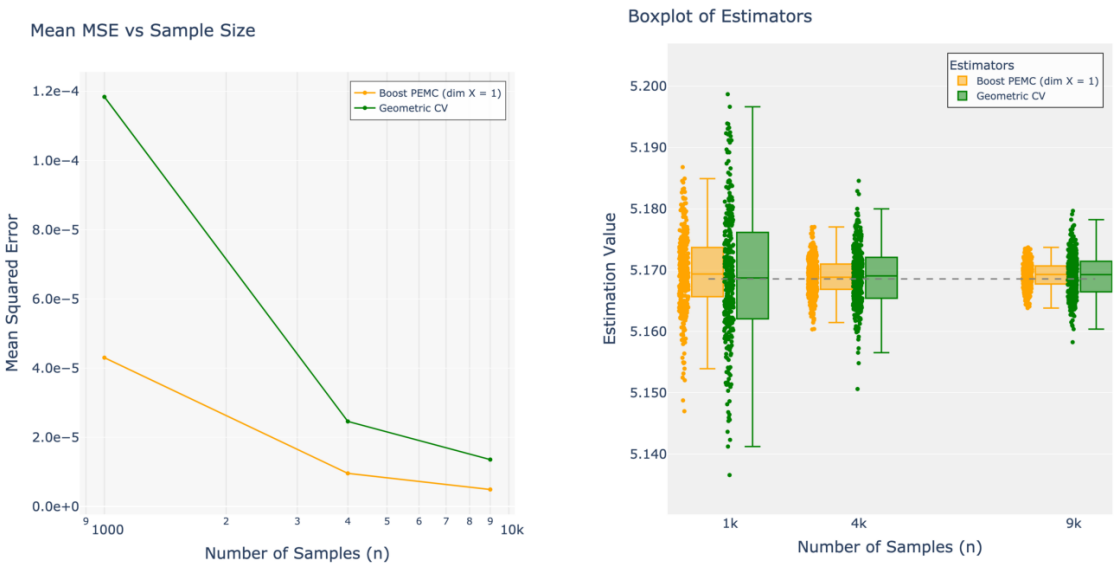

FIGURE 7. Performance of CV-based Estimators for Asian Options. Comparison of estimator performance for Boost PEMC and Geometric Control Variate (CV) based on 300 experiments. Left: Mean squared errors (MSE) plot as a function of n. Right: Corresponding boxplots of 300 estimates.

This experiment with Asian options under the GBM model serves as a controlled study to demonstrate PEMC's effectiveness as a variance reduction framework. In the following section, we go

beyond this over-simplified setting and apply PEMC to practical applications in complex derivative pricing.

Appendix C. Additional Details on the Ambulance Diversion Example

We provide more details on the ambulance diversion policy evaluation example in Section **??**. Some key ingredients are as follows: 1) Patient Arrivals: We simulate patient inflow to each hospital (A or B) according to a nonhomogeneous Poisson process with hourly rates $\lambda_{h,d}$. Specifically:

$$\text{Arrivals}(t) \sim \text{NHPP}\big(\lambda(t)\big),$$

where $\lambda(t)$ is determined by a piecewise schedule over each hour h of day d. Many of these arrivals (e.g., 25%) come by ambulance, with the remainder as walk-ins. 2) Triage Levels and Service Times: Each arriving patient is assigned a triage level $\ell \in \{1, 2, 3, 4, 5\}$ via a (possibly crisis-adjusted) multinomial distribution:

$$\Pr[L = \ell] = p_\ell \quad \text{or} \quad p_\ell^{(\text{crisis})}.$$

Service times are exponential with rate μ_ℓ, depending on the triage level ℓ. More critical levels (e.g., $\ell = 1$) have faster service rates μ_1. 3) Priority Queueing and Threshold Diversion Doctors are modeled as resources, each available over a shift [start, end]. Patients join a priority queue ordered by triage, preempting lower-priority service. A threshold τ controls ambulance diversion: if the queue Q in one hospital exceeds τ, new ambulance arrivals divert to the other hospital, incurring additional travel time Δ. Mathematically:

$$\text{divert}(\tau) = \begin{cases} 1, & \text{if } Q > \tau, \\ 0, & \text{otherwise.} \end{cases}$$

4) Mortality Function Each patient i has a time-varying risk of death. The code uses two complementary representations: 1. Direct Mortality Probability:

$$\text{mortality\_func}(x, d, a, t, B, \nu) = A_t + \frac{K_t - A_t}{\Big[1 + (3t)\exp\big(-(B + 5 - t)x + (-2 + 2.5a)t\big)\Big]^{1/(\nu + 0.25t)}},$$

where x is the wait time, d indicates diversion, a is whether the patient obtained a doctor, t is triage level, and (B, ν) control shape and shift. Constants $\{A_t, K_t\}$ reflect baseline and max risk by triage. Also, Inverse Mortality: inverse_mortality_func(u, a, t, B, ν) solves for the time at which a random uniform u indicates death. Thus each patient obtains a death_time (a form of Weibull-like distribution). A patient *dies* if its *wait time* W exceeds this random death_time. 5. We simulate day-by-day, in 4-hour shifts, updating doctors' availability and arrivals. Diversion is triggered once a queue surpasses τ. We aim to minimize total mortality over d days:

$$\min \ \mathbb{E}\Big[\sum_{\text{patients } i} \mathbf{1}\big\{\text{wait}_i > \text{death\_time}_i\big\}\Big],$$

subject to feasible travel times and doctor schedules.

To validate the proposed load-balancing rule, we develop a discrete-event simulation reflecting real-time dynamics of New York City's EMS system. Patient arrivals are modeled by a nonhomogeneous Poisson process, $\lambda(t)$, calibrated to historical data from the pandemic's peak. Upon arrival, each patient is assigned a triage level $\ell \in \{1, \ldots, 5\}$ via a multinomial distribution that shifts according to COVID surges. Service times in the emergency department follow exponential distributions with rate μ_ℓ. At each event, if the queue length at a chosen hospital exceeds a threshold, the ambulance is diverted to an alternative facility, trading off increased travel against avoiding overcrowding. By iterating this simulation over a range of thresholds and arrival scenarios, we optimize a load-balancing policy that significantly reduces hospital overloading and shortens ED wait times. Specifically, our discrete-event simulation (spanning a one-week horizon) envisions two hospitals (A and B) under

nonhomogeneous Poisson arrivals—with "normal" or "crisis" modes dictating time-varying rates—and assigns each patient a triage level from $1, 2, 3, 4, 5$. Service rates (μ_ℓ^A, μ_ℓ^B) differ by hospital and triage ℓ, while death times follow a Weibull distribution parameterized by (α, β). Our decision variable is a threshold τ, triggering ambulance diversions whenever either hospital's queue exceeds τ. We optimize τ to minimize total mortality, measured by the fraction of patients whose wait time surpasses their Weibull survival limit. Experiments show that, particularly in crisis mode, adaptive thresholding significantly reduces overloading and mortality compared to the naive "always-nearest-hospital" policy.

Let $\boldsymbol{\lambda} = (\lambda_{h,d})$ denote the matrix of arrival rates across hours and days. Patients arrive with triage levels $L \in \{1, \ldots, 5\}$ (where 1 is most critical), drawn from multinomial probabilities $\{p_\ell, p_\ell^{\text{crisis}}\}$. Within each ED, service times for triage ℓ follow exponential distributions $\exp(\mu_\ell)$, and mortality follows a logistic model: $\mathbb{P}(\text{death} \mid L = \ell, W = w) = \frac{1}{1+\exp(-(\beta_{0,\ell}+\beta_{1,\ell} w))}$. Notably, 25% of arrivals come by ambulance. If an ED's queue length Q exceeds a threshold τ, any new ambulances divert to the other ED, incurring extra travel time. In a data-adaptive version, τ is updated each shift using predicted arrival rates. All model inputs—arrival rates, triage probabilities, and service rates—come from empirical data [133], while the logistic mortality parameters $\beta_{0,\ell}$, $\beta_{1,\ell}$ are tuned (with minor per-patient perturbations) to capture heterogeneous risk profiles.

In this example, we construct the feature of

$(\tau,$ `hosp2_doctor_shift_counts`, `crisis_factor`, `hopsital1_max_patience`, `hopsital2_max_patience`, `total_number_of_patients`, `total_service_time`, `max_patience_time`, `total_death_time`, `total_number_of_patients_without_life_threatening_symptoms`, `triage1_count`, `triage2_count`, `triage3_count`, `triage4_count`, `triage5_count`$)$.

The blue features are $\boldsymbol{\theta}$ while the purple ones are $\boldsymbol{X}$. The simulation starts by first sampling the number of patients, which follows a Poisson distribution with parameter of λT, denoted by N_p. `hopsital1_max_patience` and `hopsital2_max_patience` can be simulated with max of weibull distribution, and `max_patience` can be computed by taking the max of these quantities (which has closed-form density function, i.e., maximum of Weibull's). `total_service_time` can be simulated by first sampling service time for N_p times with $\exp(\lambda_{\text{service}})$ and summing up (which has closed-form density function, i.e., sum of exponential is Gamma). `total_death_time` can be simulated by taking first sampling individual using inversion sampling from $F^{-1}(u)$, where F^{-1} :

$$F^{-1}_{t,B,\nu}(u) = \begin{cases} 0, & \text{if } u \le \text{mortality\_func}(0, t, B, \nu) \\ \infty, & \text{if } u \ge K[t] \\ -\frac{\ln\left(\left(\frac{K[t]-A[t]}{u-A[t]}\right)^{(\nu+0.25t)}-1\right)}{3t(B+5-t)} + \frac{(-2+2.5a)t}{(B+5-t)}, & \text{otherwise.} \end{cases}$$

$$K = \{1 : 1.0, 2 : 0.9, 3 : 0.05, 4 : 0.02, 5 : 0.01\}$$
$$A = \{1 : 0.6, 2 : 0.1, 3 : 0.0, 4 : 0.0, 5 : 0.0\}$$

,

$$\text{mortality\_func}(x, t, B, \nu) = A[t] + \frac{K[t] - A[t]}{(1 + (3t) \cdot \exp\left(-(B + 5 - t)x + 0.5t\right))^{\frac{1}{\nu+0.25t}}}$$

where $B \sim \text{Unif}(2.5, 3.5)$, $\nu \sim \text{Unif}(1.5, 2.5)$, and the keys of K and A are the triage levels. Triage count of each level can be simulated by firstly sampling from multi-nominal distribution with

probabilities from triage 1 to 5 as $\{0.1098, 0.2761, 0.4596, 0.1297, 0.0248\}$, and then counting the number of each category.

The quantity we are estimating is the mortality. The PEMC predictor is a random forest. The amount of training data is 10^5. During the pre-training stage, we collect the training data by the sampling procedure in Table 10.

Variable	**Distribution**
τ	Unif$\{0, 50\}$
`hosp2_doctor_shift_counts_1`	Unif$\{1, 3\}$
`hosp2_doctor_shift_counts_2`	Unif$\{1, 3\}$
`hosp2_doctor_shift_counts_3`	Unif$\{2, 6\}$
`hosp2_doctor_shift_counts_4`	Unif$\{1, 4\}$
`hosp2_doctor_shift_counts_5`	Unif$\{2, 5\}$
`hosp2_doctor_shift_counts_6`	Unif$\{1, 3\}$
`crisis`	Unif$[1, 2]$

TABLE 10. Training Data Sampling Distributions

During the evaluating stage, we fix `ciris` $= 1.25$, `hosp2_doctor_shift_counts` $= [2, 2, 4, 2, 4, 1]$ and a set of thresholds in $\{1, ..., 40\}$.

Threshold	**MSE** $\downarrow$		**MAE** $\downarrow$	
	MC	PEMC	MC	PEMC
0	10.974	4.142	2.765	1.587
4	10.749	3.899	2.751	1.500
8	10.882	4.080	2.786	1.554
12	10.243	3.609	2.638	1.466
16	9.777	3.326	2.555	1.422
20	9.535	3.266	2.497	1.416
24	9.291	3.181	2.456	1.394
28	9.236	3.192	2.445	1.403
32	9.212	3.145	2.434	1.392
36	9.175	3.120	2.427	1.382
40	9.174	3.122	2.428	1.376

TABLE 11. Error analysis for mortality estimation with varied thresholds.

APPENDIX D. FURTHER IMPLEMENTATION DETAILS OF PEMC

We discuss several further implementation details that are important for successfully applying PEMC in practice.

D.1. **Parameter Space Θ.** When determining the parameter space $\boldsymbol{\Theta}$, a key consideration is how frequently the model will be updated. To ensure the performance of PEMC, it is desirable that the training data encompass all practical scenarios the model is expected to encounter within the designated update period. For example, if one is calibrating a Heston model (6) for certain SPY ETF, on a given trading day, the calibration yields a specific set of parameters: $(S_0, \nu_0, r, \eta, \delta, \rho, \kappa) = (520, 20\%, 4.5\%, 0.04, 0.3, -0.7, 0.2)$ and the user of PEMC chooses to update the prediction model in PEMC once a month, then, based on historical data and market conditions, financial practitioners

could make reasonable guesses or confidence intervals about the bounds within which these parameters are likely to fluctuate over the coming month.

This process is conceptually similar to the selection of uncertainty sets in robust optimization [16] or distributionally robust optimization [51], where the goal is to include realizations of parameters with high likelihood. However, unlike robust optimization approaches, which often favor data-dependent uncertainty sets, our focus here is also guided by the expertise of financial engineers. The training parameter space is designed to reflect realistic and practical scenarios derived from domain knowledge, rather than being strictly driven by statistical guarantees. In applications, we construct reasonable $\boldsymbol{\Theta}$ for practical models, such as forward curves in HJM modeling [61] or local stochastic volatility 2D grids [57], among others.

D.2. **Eliminating Data Storage Overheads.** In typical ML workflows, training datasets—often costly, scarce, and carefully curated—are stored and reused extensively. By contrast, within the PEMC framework, the nature of data generation during the prediction model training in Algorithm 1, actually does not require extensive data storage. This difference stems from two main considerations. First, the volume of training data required to achieve a well-performing prediction model can be large. Storing all of it would be both expensive and unnecessary. In PEMC, the training data can be produced directly via MC simulation, ensuring an effectively unlimited supply. Second, this flexibility allows for a more efficient workflow. Data can be generated "on the fly" and processed in streaming fashion. For example, to train on a large number of N_{train} samples, one could iteratively produce small batches, train the model on these batches, and discard them after training. In this manner, we have successfully trained models in the applications using data on the order of $10^7 \smile 10^8$.

D.3. **Evaluation Metrics.** While training the NN estimator g using MSE loss is common, it is not always clear how to interpret the resulting MSE score. Unlike some well-established benchmarks (e.g., classification accuracy), there is no canonical threshold or known "good" MSE value for a given problem. This ambiguity makes it challenging to determine when the network is sufficiently trained. To address this, we can exploit the fact that g is meant to represent the conditional expectation $g = \mathbb{E}[f \mid \text{input}]$. If the network approximates this expectation well, then the sample average of $g(\boldsymbol{X})$ should be close to the sample average of $f(\boldsymbol{Y})$ over a given dataset. One practical diagnostic is to compute the Mean Absolute Relative Error (MARE) between these two averages. If $\mathbb{E}[g(\boldsymbol{X})] \approx \mathbb{E}[f(\boldsymbol{Y})]$, it provides a tangible indication that g is capturing the underlying expectation. Our empirical experience suggests this criterion is very effective in practice (a 5-1% MARE typically indicates exceptional PEMC), complementing common techniques like early stopping in ML workflows. More importantly, as the we have shown in the theory, if g closely approximates the conditional expectation, the variance reduction in PEMC is guaranteed—even if marginally—relative to standard MC.

Appendix E. Additional Details on Hyper-Parameters for Variance Swaps

Tables 12 and 13 detail the hyper-parameter setups for the NNs presented in Section **??**.

Table 12. Neural Network Architecture Parameters

CNN Branch	**Feed-forward Branch**
kernel size: 3, stride: 1, padding: 1	hidden dim: 512
max pooling (kernel: 2, stride: 2, padding: 0)	output dim: 128

2D function branch parameters	1D function branch parameters	Vector feature branch parameters
Kernel size: (1, 3)	Kernel size: 10	
Stride: (1, 3)	Stride: 3	
Padding: 0	Padding: 0	Hidden dim: 512
AvgPool2d kernel size: (2, 2)	AvgPool1d kernel size: 2	Output dim: 128
AvgPool2d stride: (2, 2)	AvgPool1d stride: 2	
AvgPool2d padding: 0	AvgPool1d padding: 0	
Feed-forward Synthesizer Parameters		
Hidden dim: 128		
Output dim: 1		

TABLE 13. Hyper-parameter setup for the neural network

APPENDIX F. REVIEW OF HJM

The HJM model directly describes the evolution of the entire forward rate curve, offering greater flexibility than traditional short-rate models. For illustration purposes, we focus on a one-factor specification with exponential volatility structure [61], though the framework readily extends to multi-factor cases.

Consider the problem of pricing a swaption. This is a contract granting its holder the right, but not the obligation, to enter into an interest rate swap at a future date. In a standard interest rate swap, one party agrees to pay a fixed rate while receiving a floating rate, and the other party does the opposite. Consider a swap with n_p fixed payment periods, each of length $\Delta t'$, starting at time t'_0 and ending at time $t'_{n_p} = t'_0 + \sum_{l=1}^{n_p} \Delta t'$. The value of this swap at time t'_0 is:

$$V_{t'_0} = C\left(R\sum_{l=1}^{n_p} B(t'_0, t'_l)\Delta t' + B(t'_0, t'_{n_p}) - 1\right),$$

where C is the notional amount (contract size), R is the fixed rate, and $B(t'_0, t'_l)$ is the discount factor from t'_0 to t'_l. A swaption provides the holder with the option to enter into this swap at t'_0. The payoff of the swaption is simply

$$\max(0, V_{t'_0}),$$

and its expectation under the risk-neutral measure gives the price of swaptions. To specify the risk neutral measure, one needs the forward rate process needed to price the bond. The time t price of a zero-coupon bond $B(t,T)$ maturing at time T is given by the negative exponential of the cumulative forward rate $f(t,u)$ as: $B(t,T) = \exp\left(-\int_t^T f(t,u)\,du\right)$, or equivalently $\frac{\partial \log B(t,T)}{\partial T} = -f(t,T)$. The HJM framework [61] models the dynamics of forward rate curve directly:

$$df(t,T) = \mu(t,T)\,dt + \sigma(t,T)^\top dW(t),$$

where $\mu(t,T)$ is the drift, $\sigma(t,T)$ is the volatility function of the forward rate, and $W(t)$ is a Brownian motion. In contrast to short-rate models (e.g., [130] or [39]), which only model the dynamics of the short-term interest rate, the HJM model directly models the dynamics of the entire term structure of interest rates [66]. The HJM model is widely used in practice because of its flexibility in modeling interest rate derivatives like swaptions and its ability to incorporate complex volatility structures [29, 5]. However, the model's generality also leads to the need for sophisticated numerical methods for simulation [61]. A key property of the HJM model is the no-arbitrage condition [61], which specifies

the drift completely by the volatility:

$$\mu(t,T) = \sigma(t,T)^\top \int_t^T \sigma(t,u)\,du. \tag{25}$$

Thus, in the HJM framework, the model is fully specified by defining the initial forward rate curve $f(0,T)$ and the structure of the volatility $\sigma(t,T)$. In our experiment we used a simple one factor HJM for illustration.