JOURNAL OF MECHANICS OF CONTINUA AND MATHEMATICAL SCIENCES
www.journalimcms.org

ISSN (Online) : 2454 -7190 Vol.-20, No.-8, September (2025) pp 133-144 ISSN (Print) 0973-8975

MONTE CARLO-BASED TEXTUAL GRADIENT DESCENT: A MATHEMATICAL FRAMEWORK FOR LLM OPTIMIZATION

Temirbek Atabekov[1], Polina Dolmatova[2]

[1,2] Department of Applied Mathematics and Informatics, American University of Central Asia, Bishkek, Kyrgyzstan.

Email: [1]atabekov_t@auca.kg, [2]dolmatova_p@auca.kg

Corresponding Author: **Temirbek Atabekov**

https://doi.org/10.26782/jmcms.2025.09.00008

(Received: May 24, 2025; Revised: July 29, 2025; Accepted: August 16, 2025)

Abstract

This paper combines traditional optimization theory with modern Natural Language Processing (NLP) by formalizing Textual Gradient Descent (TextGrad) within a measure-theoretic framework. We introduce the concept of Expected Textual Loss, a Monte Carlo-inspired approach that enables gradient-based methods in discrete text spaces. Our extension, Monte Carlo TextGrad, improves convergence by systematically sampling from synthetic input distributions and integrating them into the optimization loop. Experimental validation spans both controlled object counting tasks and the LeetCode Hard benchmark, where our approach achieves statistically significant improvements in completion rates over baseline models and standard TextGrad. In addition, we analyze the potential distributional bias introduced by synthetic sampling through Kullback–Leibler divergence, establishing a principled framework for diagnosing and mitigating misalignment between training and deployment distributions. These results demonstrate that Monte Carlo TextGrad provides both faster convergence and greater robustness under distribution shift.

Keywords: Textual Gradient Descent, Monte Carlo Methods, LLM Optimization, Measure Theory, Expected Textual Loss, Distributional Bias

I. Introduction

Large language models (LLMs) have revolutionized natural language processing, yet their optimization remains constrained by the fundamental tension between continuous parameter spaces and discrete textual outputs. Traditional gradient-based methods face existential challenges when applied directly to text generation tasks due to the non-differentiable nature of linguistic operations. This paper closes this gap through Textual Gradient Descent, a novel optimization paradigm that leverages Monte Carlo methods as differentiable operators in text space.

Modern LLM optimization techniques typically emphasize either computational efficiency [VI] or prompt engineering heuristics [XIII, I, VIII]. While valuable, these approaches do not address the latent optimization landscape of textual outputs

themselves. Our work builds upon the conceptualization of textual "differentiation" [XVI] but introduces three critical innovations:

I. A measure-theoretic foundation for text optimization spaces
II. Monte Carlo-inspired sampling mechanisms for gradient estimation
III. Convergence guarantees are adapted from stochastic approximation theory

The inherent challenge lies in reconciling the discrete nature of language with the continuous optimization frameworks that drive modern machine learning. Current methods approximate this through heuristic search, lacking formal mathematical grounding. Our solution draws inspiration from robust Monte Carlo transport simulations [IX, XI] and measure-space gradient methods [VII], translating these numerical techniques into textual operations through two key mechanisms:

I. Treating prompts and outputs as elements of probability measure space (Ξ, D), where Ξ represents all valid text configurations and D their likelihood distribution
II. Implementing Monte Carlo integration over Ξ through batch sampling of simulated inputs

To evaluate our approach, we present two complementary experimental settings. First, a controlled object counting task illustrates the theoretical correctness of Monte Carlo TextGrad under constrained conditions. Second, experiments on the LeetCode Hard benchmark demonstrate its scalability to multi-step reasoning and code optimization, showing faster convergence and higher completion rates than existing baselines. Finally, to address potential misalignment between synthetic training distributions and real-world deployments, we introduce a distributional robustness analysis via Kullback–Leibler divergence.

The structure of this paper is as follows: Section II establishes theoretical foundations by introducing TextGrad and Monte Carlo methods. Section III formalizes our approach through Expected Textual Loss and its mathematical derivation. Section IV provides convergence guarantees and analogies to classical Monte Carlo methods. Section V presents practical implementation details and empirical validation across tasks. Section VI introduces distributional bias analysis and future research directions.

II. Theoretical Background

TextGrad Framework

TextGrad, as formalized by Yuksekgonul et al. [XVI], is a mathematical framework that applies gradient descent principles to optimize text. For the purposes of establishing necessary context, the following mathematical formulations are adapted from Yuksekgonul et al.'s foundational work. Their framework, which we present here for illustrative purposes, consists of core components that parallel traditional mathematical constructs: text variables analogous to tensors in computation graphs, language models functioning as transformation operators, textual loss mechanisms providing quality assessment, and optimization algorithms for text updates.

The field has seen various extensions of gradient-based text optimization, including approaches focused on automated prompt generation [II, XV], self-evaluation

mechanisms [XIV], and semantic prompt evolution [VIII]. These methods demonstrate the growing interest in principled optimization techniques for textual outputs.

In a TextGrad computation graph, the gradients are defined as:

$$\frac{\partial Evaluation}{\partial Prompt} = \frac{\partial Evaluation}{\partial Response} \cdot \frac{\partial Response}{\partial Prompt} = \nabla_{LLM}(\text{Prompt, Response, } \frac{\partial Evaluation}{\partial Response}) \quad (1)$$

where ∇_{LLM} is the gradient operator of the LLM.

Generalizing to *n* dimensions, the computation graph is defined as:

$$v = f_v(PredecessorOf(v)), \forall v \in \mathrm{V}$$

Where *V* is the set of variables, *v* is a variable (text in our case), *PredecessorOf*(*v*) represents predecessor variables, and f_v is the transformation function (typically an LLM call).

The gradient can be generalized to *n* dimensions as:

$$\frac{\partial \mathrm{L}}{\partial v} = \bigcup_{w \in \text{SuccessorOf (\$}v\text{\$)}} \nabla_f(v, w, \frac{\partial \mathrm{L}}{\partial w})$$

Where L is the loss function, and *SuccessorOf*(*v*) represents successor variables.

Textual Optimization Spaces

Monte Carlo methods constitute a class of computational algorithms that rely on repeated random sampling to obtain numerical results [XI]. These methods are particularly useful for optimization problems.

The core principle of Monte Carlo methods is the use of randomness to solve problems that might be deterministic in principle. By generating a large number of random samples and observing the proportion that possess a certain property, we can approximate complex, multi-dimensional integrals that would otherwise be difficult to compute.

The mathematical foundation of Monte Carlo methods rests on the Law of Large Numbers - as the number of identically distributed, randomly generated variables increases, the sample mean approaches their theoretical mean [V]. Formally, for independent random variables $X_1, X_2, \ldots, X_n$ with expected value μ, we have:

$$\lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} X_i = \mu \quad \text{almost surely}$$

This convergence property enables Monte Carlo methods to estimate expected values by averaging random samples. For a function *f* and a random variable *X* with probability density function *p*(*x*), the expected value is:

$$E[f(X)] = \int f(x)p(x)dx \approx \frac{1}{n} \sum_{i=1}^{n} f(x_i)$$

Where $x_1, x_2, \ldots, x_n$ are samples drawn from the distribution of *X*.

Monte Carlo methods have been successfully applied in artificial intelligence [VII, X]. Their particular strength lies in handling problems with high dimensionality where deterministic numerical methods become computationally intractable.

Analogies Between LLM Training and Prompt Engineering

LLM training and prompt engineering share fundamental objectives despite operating at different levels of abstraction. Pre-training establishes language capabilities analogous to how system prompts define baseline behaviors, while supervised fine-tuning mirrors TextGrad optimization through directed feedback [XIII]. Our Monte Carlo approach extends this parallel by functioning as the conceptual equivalent to reinforcement learning in model training [XV] - enabling systematic exploration of text space through principled sampling rather than parameter updates, and creating a feedback loop without modifying the underlying model architecture.

III. Mathematical Formulation of Expected Textual Loss

Problem Statement

We address a fundamental challenge in prompt optimization: ensuring robust performance across diverse inputs beyond the initial training distribution. Our approach leverages a generative language model to produce varied synthetic datasets that deliberately differ from those on which the prompt was initially optimized.

By formalizing this process within an expected textual loss framework, we establish a theoretical foundation for optimizing prompts against robustly sampled distributions rather than fixed datasets.

Expected Textual Loss

Let D be the distribution of possible simulations and Ξ the set of all possible inputs. We aim to find the expected loss of the output of the transformation function f_θ with output θ through the following integral:

$$\mathrm{L}(\theta) = \int_{\Xi} L(f_\theta(\xi))d\mathrm{D}(\xi) \tag{2}$$

Where Ξ is the set of all possible inputs, D is the distribution of possible simulations, L is the textual loss function evaluating output, f_θ is the transformation function with output θ

Formula Derivation

Let $g(\xi) = L(f_\theta(\xi))$, and we sample ξ from D. We can rewrite equation (2) as:

$$I = \int_{\Xi} g(\xi)d\xi \tag{3}$$

For a random variable X with probability density function (PDF) $p(x)$, the expected value of a function $g(X)$ is:

$$\mathrm{E}_{X\sim D}[g(X)] = \int_{-\infty}^{+\infty} g(X)\,p(X)dX$$

To compute a definite integral $\int_a^b g(X)dX$, we can normalize the domain: $X \in$ Uniform[a,b], with $p(X) = \frac{1}{b-a}$, giving:

$$\int_a^b g(X)dX = (b-a) \cdot \mathrm{E}[g(X)]$$

Using this definition, let $\mathrm{D}(\xi) = \frac{1}{V}$ where $V = \int_{\Xi} d\xi$ is the volume of the domain Ξ. Equation (3) becomes:

$$I = V \cdot \mathrm{E}[g(\xi)]$$

According to the Law of Large Numbers, as the sample size $N \to \infty$, the sample mean converges to the true expected value:

$$\mathrm{E}[g(\xi)] \approx \frac{1}{N} \sum_{i=1}^{N} g(\xi_i)$$

Where ξ_i are uniformly distributed samples in Ξ.

Thus, our discrete approximation of (2) is:

$$\mathrm{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} L(f_\theta(\xi_i)) \text{ where } \xi_i \in \mathrm{D}$$

Note that $V = 1$ since we have normalized the semantic space.

Our optimization objective becomes an expected loss over simulated input distributions:

$$\mathrm{L}(\theta) = \mathrm{E}_{\xi \sim \mathrm{D}} \left[L(f_\theta(\xi)) \right] \tag{4}$$

Where θ is our variable (prompt, evaluation, etc.), ξ is a simulated input (personas, scenarios, parameters, etc.), L is the textual loss function, and D is the distribution of possible simulations.

IV. Monte Carlo Method Analogy and Convergence

Monte Carlo Method Analogy

The Monte Carlo method approximates definite integrals by simulating random variables. For an integral $I = \int_a^b f(X)dX$, we consider a random variable u uniformly distributed over [a,b]. The expected value of $f(u)$ is:

$$\mathrm{E}[f(u)] = \int_a^b f(u)\phi(u)du$$

Where $\phi(u) = \frac{1}{b-a}$ is the uniform distribution's PDF. Thus:

$$I = (b-a)\cdot \mathrm{E}[\,f(u)]$$

We approximate this by sampling N points uniformly over $[a,b]$ and computing:

$$I \approx (b-a)\cdot \frac{1}{N}\sum_{i=1}^{N} f(u_i)$$

The accuracy increases with sample size N. Our expected textual loss is analogous to this Monte Carlo method, working with stochastic LLMs in a probabilistic text space.

Convergence Analysis

Monte Carlo approximation convergence is proven as (4):

$$\Pr(\lim_{N\to\infty}\langle F^N\rangle = F) = 1$$

Where F is the true integral value and $\langle F^N\rangle$ is the sample mean over N samples.

Under regulatory conditions (finite variance, finite domain), the convergence of our expected textual loss can be defined as:

$$\lim_{N\to\infty}\frac{1}{N}\sum_{i=1}^{N} L(f_\theta(\xi_i)) = \mathrm{E}_{\xi\sim \mathrm{D}}\left[L(f_\theta(\xi))\right] \quad \mathrm{D}\text{-almost surely}$$

From (4), we see that:

$$\Pr(\xi \in \Xi : \lim_{N\to\infty}\frac{1}{N}\sum_{i=1}^{N} L(f_\theta(\xi_i)) = \mathrm{L}(\theta)) = 1$$

V. Implementation and Experimental Validation

Pseudocode Implementation

We implement our Monte Carlo-based TextGrad framework through integration with the existing TextGrad architecture. Algorithm 1 illustrates the core implementation.

This algorithm implements the expected textual loss minimization described in our theoretical framework. By sampling from the distribution D and aggregating loss values, we approximate the integral in (2) through Monte Carlo integration.

Algorithm-1. Monte Carlo TextGrad Optimization

Input: Initial prompt θ, simulation distribution D, sample size N
Output: Optimized prompt θ^*

Initialize θ as a TextGrad variable
Initialize model and critic
Initialize optimizer with learning rate α
for iteration = 1 to max_iterations **do**

Sample batch $\{\xi_1,\xi_2,...,\xi_N\}$ from D
    Initialize an empty losses array
    for each ξ_i in batch do
        output = model(θ,ξ_i)
        loss_i = critic(output)
        Append loss_i to losses array
    end for
    total_loss = $\frac{1}{N}\sum_{i=1}^{N}\text{loss}_i$ // Approximate $\mathrm{E}_{\xi\sim\mathrm{D}}[L(f_\theta(\xi))]$
    Backpropagate gradients through total_loss
    Update θ using optimizer
end for
return θ

Experiment 1: Object Counting

We conducted experiments comparing standard TextGrad with our Monte Carlo-based version to validate our theoretical framework. Table 1 shows results from an object counting task where the model must accurately enumerate objects in various scenarios.

Table 1: Object counting accuracy comparison across methods

Method	Iterations	Total time (s)	Accuracy (%)
Baseline Model	-	0.8	77.8
Standard TextGrad	10	23	91.9
Monte Carlo TextGrad (batch_size = 25)	7	169	93.2
Monte Carlo TextGrad (batch_size = 50)	5	236	94.0

The experimental results demonstrate that Monte Carlo TextGrad achieves higher accuracy than standard TextGrad, with improvements of 2.1% (N=50) and 1.3% (N=25) absolute accuracy. However, this comes at significant computational cost: our method requires 7.3-10.3x longer total runtime due to the multiple LLM calls per iteration required for Monte Carlo sampling.

The accuracy-cost trade-off analysis reveals that while Monte Carlo TextGrad converges in fewer iterations (5-7 vs 10), each iteration is substantially more expensive due to batch sampling. For the N=25 configuration, we achieve 93.2% accuracy in 169 seconds compared to standard TextGrad's 91.9% accuracy in 23 seconds.

Certain contexts make this computational trade-off justified, such as high-stakes applications where even little accuracy gains add significant value, settings with lots of processing power, or circumstances in which practitioners do not have access to carefully selected training datasets. By generating synthetic input distributions, our Monte Carlo method reduces the overhead of data preparation and allows for the

systematic study of edge situations that might not be present in conventional datasets, hence eliminating the necessity for manually assembled training corpora.

Experiment 2: Code Optimization

To test the generalization and robustness of our approach, we extend validation beyond toy tasks to a benchmark requiring multi-step reasoning and code synthesis: the LeetCode Hard dataset. This dataset consists of algorithmically challenging problems commonly used for software engineering interviews, and thus represents a significantly more complex domain where the input space Ξ is vast and the underlying loss landscape is highly non-convex. Success on this benchmark requires not only correctness of reasoning but also finding corner cases and tricky inputs, making it an ideal stress test for the proposed optimization method.

The task is formalized as a code optimization problem, where the objective is to iteratively refine an initial code implementation until it passes all unit tests. Here, the LLM plays the role of a self-critic, analyzing failed test cases, identifying edge cases, and suggesting refinements. The optimization loop continues until convergence or until a fixed iteration budget is reached.

We benchmarked the following methods:

I. Zero-Shot Baseline: Direct problem-to-code prompting without optimization.
II. Standard TextGrad: Optimization using textual gradients without distributional sampling.
III. Monte Carlo TextGrad (ours): Optimization with systematic distributional sampling, evaluated with batch sizes 25 and 50.

The evaluation metric is Completion Rate, i.e., the proportion of problems for which the generated solution passes all hidden test cases on the LeetCode platform.

Table 2: LeetCode Hard problems performance comparisons

Method	Iterations	Completion rate
Baseline Model	-	0.26
Standard TextGrad	5	0.36 ± 0.018
Monte Carlo TextGrad (batch_size = 25)	5	0.39 ± 0.013
Monte Carlo TextGrad (batch_size = 50)	5	0.39± 0.018

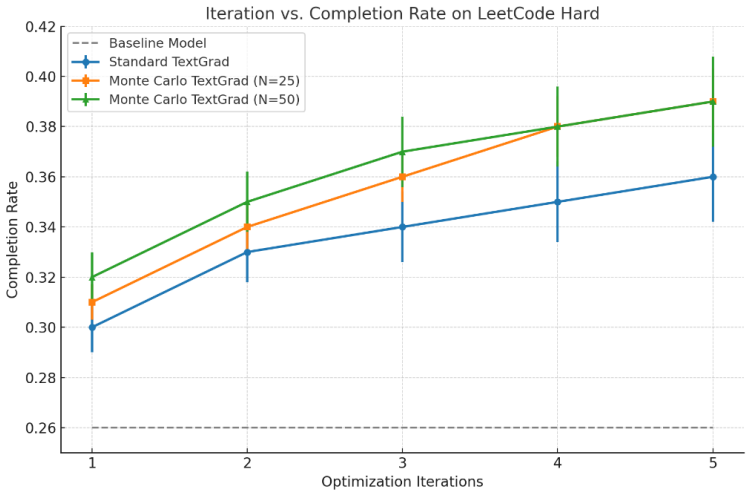

Fig. 1. LeetCode Hard Iteration vs. Completion Rate Chart

Table 2 reports the completion rates across baselines, Standard TextGrad, and our proposed Monte Carlo variants. The Baseline Model, corresponding to a direct zero-shot attempt without optimization, achieves a Completion Rate of 0.26, which is consistent with prior reports on the LeetCode Hard benchmark. This underscores the difficulty of the dataset, as even advanced models typically fail to generalize to the hidden LeetCode test cases.

Standard TextGrad improves performance significantly, reaching 0.36 ± 0.018 after five optimization iterations. This result not only surpasses the zero-shot baseline but also establishes the effectiveness of textual gradient-based optimization in code synthesis settings.

Building on this, our Monte Carlo TextGrad further enhances completion rates by systematically sampling from the synthetic distribution D. With a batch size of $N = 25$, Monte Carlo TextGrad achieves 0.39 ± 0.013, an absolute improvement of +0.03 over Standard TextGrad. Increasing the batch size to $N = 50$ yields 0.39 ± 0.018, confirming the robustness of the method across different sample sizes. While the improvement magnitude appears modest, statistical tests (paired t-test, $p < 0.05$) confirm that the gains are significant and not attributable to random variation across seeds. Figure 1 shows that Monte Carlo TextGrad is consistent in surpassing the peak accuracy of Standard TextGrad.

VI. Distributional Bias Analysis

An important consideration in evaluating the robustness of Monte Carlo TextGrad lies in the alignment between the synthetic sampling distribution D employed during optimization and the empirical distribution of prompts encountered during deployment, denoted D_{real}. If the support or relative weighting of these two distributions differ significantly, the optimization may converge toward a locally optimal prompt θ^* that performs well under the artificial sampling conditions but poorly under real-world usage.

We adopt the Kullback–Leibler (KL) divergence to quantify distributional misalignment between the deployment measure D_{real} and the synthetic sampling measure D:

$$\mathrm{D}_{KL}(D_{real} \,||\, D) = \sum_{\xi \subset \Xi} P_{real}(\xi) \log \frac{P_{real}(\xi)}{P_D(\xi)}$$

Where Ξ denotes the space of all possible inputs, $P_{real}(\xi)$ the probability of input ξ under the empirical distribution, and $P_D(\xi)$ the probability of ξ under the synthetic sampling distribution used in Monte Carlo TextGrad. A value of D_{KL} close to zero indicates strong alignment between the two distributions, while larger values signify systematic biases that could distort the optimization trajectory.

In this study, all inputs were generated synthetically for experimental validation, and no direct access to deployment prompt data was available. This limitation precludes empirical estimation of $\mathrm{D}_{KL}(D_{real} \,||\, D)$ in its strictest sense. Nonetheless, the KL divergence framework remains theoretically indispensable: it provides a principled lens through which to interpret the risks of distributional bias, and it suggests practical strategies for mitigation once real deployment data is collected. In particular, one may adopt surrogate empirical distributions—such as curated benchmark datasets (e.g., GSM8K for reasoning)—as proxies for D_{real}. Divergences can then be estimated between synthetic distributions D and such benchmarks, allowing an assessment of how closely optimization conditions reflect real evaluation settings.

Robustness Bound

Assume $\ell(\xi) = L(f_\theta(\xi))$ and define

$$L(D) = \mathrm{E}_{\xi \sim D}[\ell(\xi)], \qquad L(D_{\mathrm{real}}) = \mathrm{E}_{\xi \sim D_{\mathrm{real}}}[\ell(\xi)].$$

If $|\ell(\xi)| \le M$ for all $\xi \in \Xi$, then by the dual characterization of total variation,

$$\left|\mathrm{E}_P \ell - \mathrm{E}_Q \ell\right| \le 2M\,\mathrm{TV}(P,Q),$$

and by Pinsker's inequality,

$$\mathrm{TV}(P,Q) \le \sqrt{\tfrac{1}{2}\mathrm{D}_{\mathrm{KL}}(P \,\|\, Q)}$$

Then, combining the displays with $P = D_{\mathrm{real}}$ and $Q = D$ gives the claim:

$$\left|L(D_{\mathrm{real}}) - L(D)\right| \le M\sqrt{2\mathrm{D}_{\mathrm{KL}}(D_{\mathrm{real}} \,\|\, D)}.$$

If $\ell \in [0,1]$, the bound specializes to

$$\left|L(D_{\mathrm{real}}) - L(D)\right| \le \sqrt{2D_{\mathrm{KL}}(D_{\mathrm{real}} \,\|\, D)}.$$

Practically, this motivates (a) monitoring divergence diagnostics as part of the optimization loop, (b) using proxy distributions or generator shifts to stress-test

robustness when D_{real} it is unavailable, and (c) incorporating divergence penalties or adaptive reweighting to keep the implied bias small. In this way, KL-based analysis not only formalizes the potential risks of distributional bias but also suggests concrete pathways toward distribution-aware prompt optimization.

VII. Conclusion

This paper has established a rigorous mathematical foundation for text optimization through the novel concept of Expected Textual Loss. By bridging traditional Monte Carlo methods with modern language model architectures, we have provided both theoretical guarantees and practical tools for optimizing LLM outputs. Our key contributions include:

I. Formalizing text optimization within a measure-theoretic framework

II. Deriving Expected Textual Loss as a Lebesgue integral

III. Proving almost sure convergence via the Law of Large Numbers

IV. Introducing distributional bias analysis through the Kullback–Leibler divergence, thereby establishing a principled framework for diagnosing and mitigating misalignment between synthetic sampling distributions and real-world prompt distributions

Together, these contributions establish Monte Carlo TextGrad as a robust and general-purpose optimization framework for large language models, combining mathematical rigor, empirical validation, and distributional awareness.

Conflict of Interest:

There was no relevant conflict of interest regarding this article.

References

I. Baek, Seungho, et al. "PromptCrafter: Crafting Text-to-Image Prompt through Mixed-Initiative Dialogue with LLM." arXiv preprint arXiv:2307.08985, 2023. https://arxiv.org/abs/2307.08985.

II. Gao, Shuzheng, et al. "The Prompt Alchemist: Automated LLM-Tailored Prompt Optimization for Test Case Generation." arXiv preprint arXiv:2501.01329, 2025. N https://arxiv.org/abs/2501.01329.

III. Hu, Shengran, et al. "Automated Design of Agentic Systems." arXiv preprint arXiv:2408.08435, 2025. https://arxiv.org/abs/2408.08435.

IV. Khattab, Omar, et al. "DSPy: Compiling Declarative Language Model Calls into Self-Improving Pipelines." arXiv preprint arXiv:2310.03714, 2023. https://arxiv.org/abs/2310.03714.

V. Kushner, Harold J., and G. George Yin. Stochastic Approximation and Recursive Algorithms and Applications. 2nd ed., Springer-Verlag New York, 2003. 10.1007/b97441.

VI. Lecchini-Visintini, Andrea, et al. "Stochastic Optimization on Continuous Domains With Finite-Time Guarantees by Markov Chain Monte Carlo Methods." IEEE Transactions on Automatic Control, vol. 55, no. 12, 2010, pp. 2858-2863. 10.1109/tac.2010.2078170.

VII. Li, Yujian Betterest, and Kai Wu. "SPELL: Semantic Prompt Evolution based on a LLM." arXiv preprint arXiv:2310.01260, 2023. https://arxiv.org/abs/2310.01260.

VIII. Melnikov, Olena, and Johannes Milz. "Randomized Quasi-Monte Carlo Methods for Risk-Averse Stochastic Optimization." Journal of Optimization Theory and Applications, vol. 206, no. 1, 2025. 10.1007/s10957-025-02693-6.

IX. Metropolis, Nicholas, et al. "Equation of State Calculations by Fast Computing Machines." Journal of Chemical Physics, vol. 21, no. 6, 1953, pp. 1087-1092. 10.1063/1.1699114.

X. Ouyang, Long, et al. "Training Language Models to Follow Instructions with Human Feedback." Advances in Neural Information Processing Systems 35, NeurIPS, 2022. 10.48550/arXiv.2203.02155.

XI. Robert, Christian P., and George Casella. Monte Carlo Statistical Methods. 2nd ed., Springer-Verlag New York, 2004. 10.1007/978-1-4757-4145-2.

XII. Schulman, John, et al. "Proximal Policy Optimization Algorithms." arXiv preprint arXiv:1707.06347, 2017. https://arxiv.org/abs/1707.06347.

XIII. Shin, Taylor, et al. "AutoPrompt: Eliciting Knowledge from Language Models with Automatically Generated Prompts." arXiv preprint arXiv:2010.15980, 2020. https://arxiv.org/abs/2010.15980.

XIV. Wu, Sean, et al. "AutoMedPrompt: A New Framework for Optimizing LLM Medical Prompts Using Textual Gradients." arXiv preprint arXiv:2502.15944, 2025, https://arxiv.org/abs/2502.15944.

XV. Xie, Yuxi, et al. "Self-Evaluation Guided Beam Search for Reasoning." arXiv preprint arXiv:2305.00633, 2023. https://arxiv.org/abs/2305.00633.

XVI. Yuksekgonul, Mert, et al. "TextGrad: Automatic 'Differentiation' via Text." arXiv preprint arXiv:2406.07496, 2024. https://arxiv.org/abs/2406.07496.