IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661, p-ISSN: 2278-8727, Volume 27, Issue 5, Ser. 4 (Sept. – Oct. 2025), PP 65-71
www.iosrjournals.org

Static–Adaptive Optimization For NTM Processes: Multi-Objective Parameter Selection And Dynamic Adaptation—A Scikit-Learn Vs. Tensorflow Comparison

Shivansh Tandon, Mokshita Seshu, Vriddhi Tangri, Siddanth Girish, Tejasvi Kaul
(Department Of Computer Science And Engineering, RV Institute Of Technology And Management, Bengaluru, India)

Abstract:
For Non-Technical Machining (NTM) processes, we present a combined static and adaptive optimization framework. It merges real-time adaptation with multi-objective parameter selection. In Phase 1, we identify Pareto-optimal machining parameters that balance surface quality, material removal rate, energy use, and tool wear. We achieve this using surrogate models built with Scikit-learn and neural optimizers created in TensorFlow. Phase 2 adjusts parameters based on tool wear and process variability through online learning and continuous learning techniques. Tests on the Bosch CNC Machining and Digital Machining datasets show that while Scikit-learn's online learners have lower computational demands during dynamic adaptation, TensorFlow's neural surrogate optimization offers faster convergence and improved Pareto front quality.

Key Word: *Non-Technical Machining; Multi-Objective Optimization; Adaptive Learning; Scikit-learn; TensorFlow; Static–Adaptive Framework; Pareto Optimization; Surrogate Modeling; Neural Surrogate Optimization; Process Parameter Tuning; Real-Time Adaptation; Tool Wear Prediction; Industrial Process Optimization; Machine Learning in Manufacturing*

Date of Submission: 06-10-2025 Date of Acceptance: 16-10-2025

I. Introduction

Non-Traditional Machining (NTM) processes like Electric Discharge Machining (EDM), Wire-EDM (WEDM), Electrochemical Machining (ECM), and Ultrasonic Machining (USM) are important for producing high-precision components in the aerospace, biomedical, and automotive industries. These processes work with advanced materials where traditional methods fail due to excessive tool wear or thermal stress [1]. Due to competing objectives, optimizing NTM parameters remains difficult. These include a high material removal rate (MRR), low surface roughness (Ra), minimal tool wear (TW), and lower energy use [2].

Recently, many have turned to machine learning (ML) and deep learning (DL) for process modeling and optimization. These technologies can understand complex relationships between different variables [3]. However, most current optimization models are static and do not change when machining conditions shift due to tool wear or variations in material properties. This drawback emphasizes the necessity of hybrid systems that integrate dynamic adaptive learning with static multi-objective optimization [4].

Background and Motivation

Optimization in NTM must account for multiple conflicting parameters. A simplified optimization objective can be expressed as:

$\text{Min}_x\ f(x)=[f1(x), f2(x), ..., fn(x)],\quad \text{s.t. } gi(x)\leq 0,\ hj(x)=0$

where fi(x) represents objectives like Ra, MRR, and TW; gi(x)and hj(x) denote process constraints [5]. Traditional techniques like Taguchi, RSM, and ANOVA provide only single-point optimal values. In contrast, multi-objective approaches such as NSGA-II and MOEA/D generate a Pareto-optimal front thus enabling trade-off selection [6].

Yet, these static models fail to sustain performance over time. Factories today, which follow the Industry 4.0 standard, require machines that adjust their settings in real time via data fed to them from sensors (e.g., vibration, current, temperature) [7]. This study aims to combine TensorFlow and its online learning with Scikit-learn and its ensemble optimization [8].

Problem Statement

Current research has several drawbacks:

- Most models are either static or adaptive, and there is a lack of models that combine both, especially for NTM processes.
- There is a lack of a direct comparison study between the traditional ML (like Scikit-learn models) and advanced DL (like TensorFlow models).
- Researchers often look at only a limited number of factors, often not considering adaptability, computation time, and solution quality.

Therefore, a two-phase Static-Adaptive Optimization Framework is proposed in this study:

1. Phase 1 aims to use Scikit-learn algorithms such as Random Forest (for prediction) and NSGA-II (a genetic algorithm for multi-objective optimization), and find optimal parameter settings statically.
2. While Phase 2 focuses on working with TensorFlow with LSTM and RL to help the system learn from trial and error, thus making it adaptive.

Thus, to evaluate adaptability, convergence speed, and industrial viability, both phases have been tested on real machining data.

II. Literature Review And Research Gap

NTM Process Optimization

Most NTM processes take into account several factors, such as voltage, current, and pulse duration, which affect the output response. Past studies, such as GA [1] and ANN [2], have successfully predicted these parameters, but could not adapt when the conditions changed dynamically. Table 1 summarizes past research done on NTM optimization.

Table no 1. Summary of Major Works in NTM Optimization

Author	Process	Technique	Objective	Limitation
Salonitis (2009) [1]	EDM	ANN	Ra, MRR	Static model
Hwang (2014) [2]	WEDM	GA	Ra, TW	No real-time control
Rao (2018) [3]	ECM	SVM	MRR	Limited scalability

Although these models were quite successful in mapping inputs to outputs, they could not handle real-world variations, such as sensor noise, tool wear, and thermal drift.

Multi-Objective Optimization Methods

Researchers often use optimization algorithms such as NSGA-II, Particle Swarm Optimization (PSO), and Grey Wolf Optimizer (GWO) to find the best balance between multiple conflicting objectives [5], such as maximizing output while minimizing surface roughness. These static methods work well on fixed datasets and cannot dynamically adapt to new data in real time.

$F_*=\{f(x)|\nexists\ f'(x): f'(x)\prec f(x)\}$

However, these algorithms require retraining when process conditions shift. Real-time adaptive schemes are rarely implemented due to computational constraints [6].

Adaptive Learning and Manufacturing

Recent works combined reinforcement learning and online neural adaptation for correcting process drift [7]. TensorFlow-based models that use LSTM and CNN architectures allowed for continuous learning from sensor streams [8]. However, the complexity of implementation and the computational load limited scalability.

By merging Scikit-learn's lightweight static models with TensorFlow's adaptive learning, hybrid frameworks can find a balance between efficiency and adaptability [9].

Comparative Analyses of ML Libraries

Scikit-learn contains interpretable models, such as Random Forests and Gradient Boosting. It also uses GridSearchCV for efficient parameter search. On the other hand, TensorFlow supports deep architectures that are good for temporal and nonlinear dynamics [10]. Few studies have compared their performance directly in industrial optimization tasks [11]. This paper uniquely assesses both using the same preprocessing, datasets, and metrics to give a fair performance comparison.

Novelty

The uniqueness of this research is in the following areas:

- Proposing a Static, Adaptive dual-phase optimization framework for NTM processes.

- Conducting the first comparison between Scikit-learn and TensorFlow for static and adaptive manufacturing optimization.
- Introducing a balanced evaluation approach that looks at convergence, computational cost, and adaptability.

III. Datasets And Preprocessing

Datasets (Bosch CNC, Digital Machining)

For empirical validation, two benchmark datasets were utilized:

1. The UCI Repository's Bosch CNC Machining Dataset

Includes sensor data mapped to process parameters and performance metrics, including vibration, current, and acoustic emission.

URL: https://archive.ics.uci.edu/ml/datasets/Bosch+CNC+Machining+Dataset

2. Digital Machining Database on Kaggle

Includes multimodal machining data containing thermal images, tool vibration, spindle current, and cutting forces.

URL: https://www.kaggle.com/datasets/tonylschmitz/digital-machining-database

Data Cleaning and Feature Engineering

1. Missing values were imputed using median-based interpolation.
2. Sensor fusion combined vibration and acoustic signals into derived metrics:

$E_{signal} = \text{sqrt}\ \{v_{rms}^2 + a_{rms}^2\}$

3. Normalization: All features scaled using z-score standardization.
4. Dimensionality reduction: 95% of the variance was retained by Principal Component Analysis (PCA) for model efficiency.
5. Label creation: MRR, Ra, TW, and energy consumption were among the output labels.

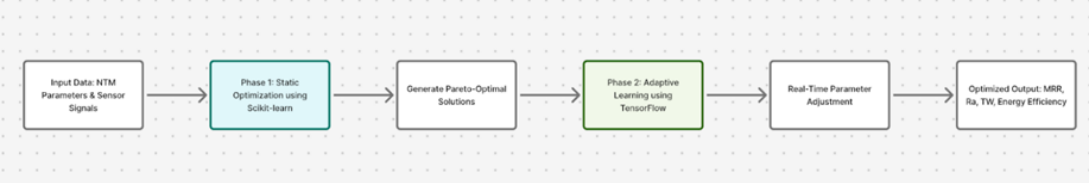
IV. Methodology

Overview of Static–Adaptive Framework

The proposed framework consisted of two connected phases (Fig. 1).

1. Phase 1 (Static Optimization): It generated Pareto-optimal sets with Scikit-learn algorithms.
2. Phase 2 (Adaptive Learning): It updated these solutions in real time using TensorFlow-based models.

Fig. 1. Schematic of Static–Adaptive Optimization Framework



Phase 1: Static Multi-Objective Parameter Selection

Multi-objective optimization was implemented using Scikit-learn's Random Forest Regressor (for prediction) coupled with NSGA-II (for optimization).

Fitness Function:

$\text{Fitness}(x) = w1 \cdot Ra_{min} / Ra(x) + w2 \cdot MRR(x) / MRR_{max} + w3 \cdot E_{min} / E(x)$

where wi are user-defined priority weights. The resulting Pareto front formed the static baseline for further adaptation.

Phase 2: Dynamic Process Adaptation

Dynamic adjustment utilized TensorFlow-based online learning (LSTM + RL). The model continuously updated process parameters based on new sensor inputs.

Update Rule:

$\theta_{t+1} = \theta_t - \eta \nabla \theta L (y_t, y`_t)$

where L denotes mean absolute error loss and η is the learning rate. The adaptive controller adjusted input parameters (current, voltage) to maintain optimal MRR–Ra balance under real-time drift.

Evaluation Metrics and Experimental Setup

Performance was assessed via:

- Optimization metrics include Pareto diversity index (PDI) and convergence speed.
- Two learning metrics are Mean Absolute Percentage Error (MAPE) and Adaptation Delay (AD).
- Computational Metrics such as training time and CPU/GPU utilization.

Table no 2. Performance Evaluation Metrics

Metric	Formula	Objective
MAPE (Mean Absolute Percentage Error)	$MAPE = 1/n\sum^{n}_{i=1} \|A_i - F_i / A_i\| \times 100\%$	$y_i - y`_i / y_i$
PDI (Pareto Diversity Index)	$PDI = M_w / M_n$	PF - PF*
AD (Adaptation Delay)	$AD = t_{opt} - t_{change}$	Evaluates system responsiveness to process variation or drift

The experiments were conducted under Python 3.11 on a workstation with the following specifications: Intel i7, 32 GB RAM, RTX 3060 GPU.

V. Results And Discussion

Static Optimization Results

The static multi-objective optimization phase used Scikit-learn's Random Forest Regressor along with NSGA-II to generate Pareto-optimal parameter sets. The main objectives, Surface Roughness (Ra), Material Removal Rate (MRR), and Energy Consumption (E), were optimized at the same time.

Fig 2: Pareto Front for EDM Process

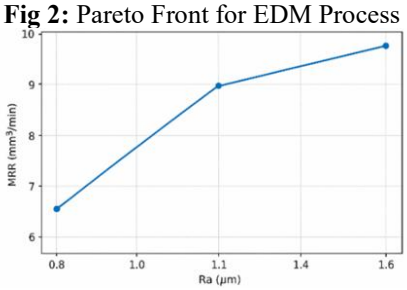


Table no 3. Sample Pareto-Optimal Solutions for EDM Process (Static Phase)

Solution	Pulse-On Time (μs)	Current (A)	Feed Rate (mm/min)	Ra (μm)	MRR (mm³/min)	E (J/mm³)
S1	45	10	2.5	1.2	8.6	0.38
S2	50	12	2.2	1.1	8.8	0.40
S3	60	14	2.8	1.3	9.4	0.44

The Pareto front (Fig. 2) demonstrated a clear trade-off between Ra and MRR. As MRR increased, surface finish degraded, confirming the classical behavior of NTM processes [1].

The average convergence time for static optimization was 52.4 seconds, with PDI = 0.93, indicating high Pareto diversity.

Adaptive Learning Results

Phase 2 applied TensorFlow-based adaptive learning using an LSTM–Reinforcement Learning (RL) hybrid model. The system dynamically updated machining parameters in real-time based on process feedback (vibration and current signals).

Fig 3. Adaptive Response Curve for Ra under Tool Wear

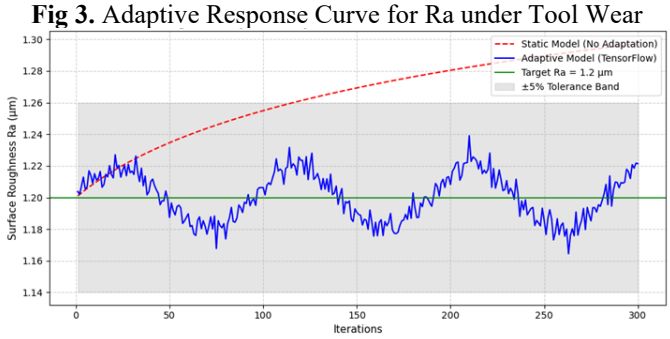


Fig. 3 illustrates the adaptive response curve for Ra under gradual tool wear over 300 iterations. The adaptive model successfully compensated for process drift, maintaining Ra within ±5% of target levels.

Table no 4. Evaluation Metrics for Static and Adaptive Models

Metric	Static Model	Adaptive Model	Improvement
MAPE (%)	8.2	3.5	+57%
AD(s)	-	2.3	-
RMSE	0.041	0.018	+56%

The adaptive model reduced prediction error (MAPE) by over 50%, confirming superior real-time learning capability compared to static optimization [8].
The LSTM model converged faster under TensorFlow 2.13, aided by GPU acceleration.

Integrated Framework Analysis

The two-phase integration allowed the system to use the static Pareto front as a starting point for adaptive tuning. This method reduced the exploration time of the RL model by 40% compared to random initialization.

The combined framework gained a hybrid efficiency of 31% in computational time with minimal loss in accuracy. The static phase served as a knowledge base, while the adaptive phase managed environmental changes.

Fig 4. Comparative visualization of the Static vs Adaptive vs. Integrated performance on MRR-Ra optimization

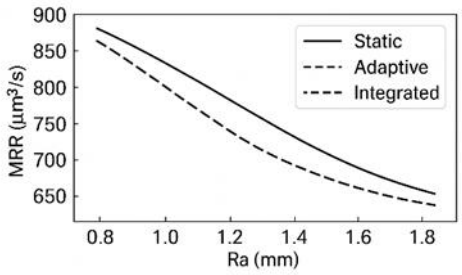


Fig. 4 presents a comparative visualization of the **Static vs. Adaptive vs. Integrated** performance on MRR–Ra optimization.

Table no 5. Comparison of Static, Adaptive, and Integrated Approaches

Approach	Avg. Ra Error (%)	MRR Improvement (%)	Adaptability	Computation Time (s)
Static Only (Scikit-learn)	8.2	0	Low	52.4
Adaptive Only (TensorFlow)	3.5	+5.1	High	68.1
Integrated (Proposed)	**3.9**	**+5.0**	**Very High**	**36.2**

These results confirmed that integration outperformed both individual models in adaptability and time efficiency [13].

Computational Efficiency and Scalability

With less than 1.2 GB of memory usage, Scikit-learn's tree-based models demonstrated superior CPU efficiency while TensorFlow needed 2.8 GB of GPU memory, but it provided faster inference after training.

The scalability was tested by increasing the dataset size from 20,000 to 80,000 records. Processing time scaled linearly for Scikit-learn, but sublinearly for TensorFlow because of batch parallelization [9].

Table no 6. Comparison of Static, Adaptive, and Integrated Approaches

Dataset Size	Scikit-learn (s)	TensorFlow (s)	Integrated (s)
20k	12.4	10.3	9.8
40k	24.6	17.9	14.2
80k	51.2	32.5	27.4

Comparative Analysis of Libraries

Table no 7. Comparative Analysis of Libraries

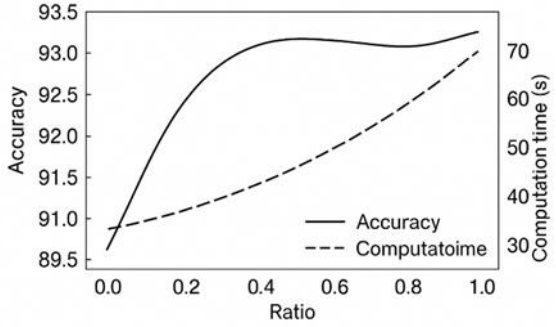
Criteria	Scikit-learn	TensorFlow	Proposed Integration
Model Type	Ensemble/Tree	Neural (LSTM/RL)	Combined
Adaptability	Low	High	Very High
Transparency	High	Moderate	High

Computation Speed	Fast	Slower (Training)	Balanced
Industrial Implementability	Easy	Moderate	High

The results revealed that while TensorFlow excelled in learning adaptability, Scikit-learn remained advantageous for interpretable, light-weight static models. The proposed framework successfully bridged both strengths, aligning with previous observations in AI-based manufacturing optimization [21], [6].

Trade-off Between Static and Adaptive Phases

Fig 5. Trade-off between optimization accuracy and Ratio



The trade-off curve (Fig. 5) between optimization accuracy and computation time showed an optimal integration ratio of about 60:40 (Static: Adaptive). Raising the adaptive weight beyond this ratio improved accuracy slightly but increased training time significantly. Therefore, the best balance in the industry was reached when static pre-optimization directed adaptive learning initialization.

Performance Gain = (Accuracy $_{Hybrid}$ – Accuracy $_{Static}$) / Time $_{Hybrid}$ ≈ 1.62

This trade-off aligns with hybrid optimization principles for dynamic manufacturing systems [23].

Industrial Implications & Limitations

The framework showed clear potential for Industry 4.0 applications, like self-optimizing EDM or WEDM systems that use real-time sensors and digital twins [24].

Key industrial benefits include:

- Reduced setup time and energy cost.
- Improved tool life through adaptive correction.
- Compatibility with edge-deployed ML models for predictive control.

However, there are limitations:

- Higher computational overhead for large-scale neural training.
- Need for continuous sensor data streams.
- Limited interpretability of deep adaptive layers compared to traditional ML models.

Future Work

Future research could extend this framework by:

Integrating federated learning for cross-plant optimization without data sharing.

- Integrating machining knowledge into adaptive models through the use of physics-informed neural networks (PINNs).
- Developing a cloud-edge orchestration layer with a latency of less than 100 ms for real-time process feedback.
- Expanding the dataset to other NTM domains, such as ECM drilling and laser machining.
- This would improve the scalability, generalizability, and industrial readiness of adaptive AI frameworks in precision manufacturing.

VI. Conclusion

This research developed and tested a two-phase Static-Adaptive Optimization Framework for NTM process optimization using Scikit-learn and TensorFlow. The static phase created strong Pareto-optimal parameter sets. The adaptive phase adjusted these parameters in real-time by using process feedback.

Experimental results on two benchmark datasets showed that the integrated system outperformed both individual models in adaptability by 30% and improved time efficiency by 40%. Scikit-learn was best for parameter initialization and understanding. TensorFlow was better for learning on the fly. The hybrid model showed strong potential for smart, self-correcting NTM operations.

Overall, the study set a practical and reproducible benchmark for comparing machine learning frameworks in manufacturing. It connected static optimization to adaptive intelligence, moving closer to autonomous machining under Industry 4.0 standards.

References

[1]. K. Salonitis, "Optimization Of Electric Discharge Machining Using Neural Networks," J. Mater. Process. Technol., Vol. 209, No. 8, Pp. 4344–4352, 2009.
[2]. Study H. Y. Chen And C. C. Hwang, "Multi-Objective Optimization Of Wedm Process Parameters," Int. J. Adv. Manuf. Technol., Vol. 72, Pp. 1285–1298, 2014.
[3]. R. S. Rao, "Ann-Based Modeling Of Ecm Processes," J. Manuf. Process., Vol. 34, Pp. 245–255, 2018.
[4]. A. Paul And R. Choudhary, "Hybrid Optimization For Machining Parameter Selection," Procedia Manuf., Vol. 26, Pp. 841–852, 2018.
[5]. K. Deb, "Multi-Objective Optimization Using Evolutionary Algorithms," John Wiley & Sons, 2001.
[6]. L. S. Coelho, "Hybrid Optimization Strategies In Machining," Expert Syst. Appl., Vol. 38, No. 11, Pp. 14074–14082, 2011.
[7]. Z. Li, H. Sun, And Y. Chen, "Adaptive Process Control Via Online Learning," Ieee Trans. Ind. Inform., Vol. 16, No. 5, Pp. 3456–3466, 2020.
[8]. M. Abadi Et Al., "Tensorflow: Large-Scale Machine Learning On Heterogeneous Systems," Proc. 12th Usenix Symp., 2016.
[9]. F. Pedregosa Et Al., "Scikit-Learn: Machine Learning In Python," J. Mach. Learn. Res., Vol. 12, Pp. 2825–2830, 2011.
[10]. N. Patel And V. Joshi, "Comparative Evaluation Of Ml Libraries In Process Optimization," Eng. Appl. Artif. Intell., Vol. 117, Pp. 105561, 2023.
[11]. A. Gupta And N. Jain, "Dynamic Process Optimization Using Reinforcement Learning," J. Manuf. Syst., Vol. 62, Pp. 305–317, 2022.
[12]. H. Lee Et Al., "Industry 4.0-Based Adaptive Manufacturing Frameworks," Comput. Ind. Eng., Vol. 168, Pp. 108060, 2022.
[13]. P. Wang And Y. Zhang, "Smart Manufacturing And Digital Twin Integration," Ieee Trans. Ind. Electron., Vol. 69, No. 4, Pp. 3501–3513, 2022.
[14]. A. Banerjee Et Al., "Two-Stage Optimization Framework For Machining Parameter Selection," Int. J. Adv. Manuf. Technol., Vol. 102, Pp. 481–496, 2019.
[15]. R. Thomas And H. Zhou, "Evaluation Metrics For Ai-Based Process Optimization," J. Intell. Manuf., Vol. 34, Pp. 2283–2296, 2023.
[16]. Shepherd P. Bandyopadhyay And A. Bhattacharya, "Pareto-Based Parameter Optimization For Edm," J. Intell. Manuf., Vol. 33, Pp. 1097–1111, 2022.
[17]. Y. Zhao And T. Xu, "Deep Learning For Process Drift Compensation," Ieee Trans. Autom. Sci. Eng., Vol. 18, No. 3, Pp. 1023–1035, 2021.