TurboGrad: An Open-source Differentiable Performance Model for Aeroengine Condition Monitoring

Daniel Cisneros Acevedo[1], Marta Ribeiro [1], Ingeborg de Pater [1], Manuel Arias Chao [1, 2], and Tim Rootliep [3]

[1] *Faculty of Aerospace Engineering, Delft University of Technology, Delft, Netherlands*

[2] *Institute of Data Science, Zurich University of Applied Sciences, Winterthur, Switzerland*

[3] *KLM Engine Services, Schiphol-Oost, Netherlands*

ABSTRACT

Combining physics-based and data-driven models for aeroengine condition monitoring has attracted increasing research interest in Prognostics and Health Management. Hybrid methods that incorporate physics-based models into deep neural networks typically rely on pre-trained surrogate models of the engine performance model, which serve as a differentiable proxy during training. However, constructing such surrogates requires extensive exploration of the parameter space to generate representative datasets, resulting in a rapidly increasing computational burden as the number of model parameters grow. To address this limitation, we present TurboGrad, an open-source differentiable aeroengine performance model that reformulates the Gas Turbine Simulation Program (GSPy) in PyTorch. Because the performance model is tracked as a computation graph, gradients with respect to any model parameter follow directly via backpropagation. We compared TurboGrad against GSPy for a single-spool turbojet, finding relative errors within 0.3%. Furthermore, we demonstrate gradient-based estimation of compressor and turbine efficiencies, converging to the ground truth after 30 epochs. TurboGrad is open-source and provides a differentiable foundation for integrating physics-based aeroengine models directly into deep learning pipelines.

1. INTRODUCTION

Condition monitoring of aeroengines has been pursued since their early deployment, motivated by the need to reduce maintenance costs while ensuring safe and reliable operation. In particular, gas path analysis (GPA) is a research field that concerns itself with detection and isolation of aeroengine faults using physics-based performance models. These models relate observable quantities, such as temperatures, pressures, and rotational speeds, to unobservable parameters that represent the health condition of critical components within the engine (Volponi & Tang, 2025). Hence, estimating component condition requires fitting the physical model to condition monitoring data. This fitting procedure is, in effect, an inverse problem (Vu, Razakarivony, Marnissi, & Nocture, 2024).

Several factors complicate this estimation process. First of all, engine performance is highly non-linear with respect to the health parameters and the operating conditions (Li et al., 2011). Secondly, model incompleteness, instrumentation bias, and sensor noise introduce further uncertainty in the predictions (Chao, Lilley, Mathé, & Schloßhauer, 2015). Third, when the number of parameters being monitored exceeds the number of gas-path sensors, the problem becomes underdetermined, i.e., multiple solutions may exist (Borguet, 2012).

Researchers have consequently explored physics-informed machine learning approaches to aeroengine condition monitoring (Fentaye, Baheta, Gilani, & Kyprianidis, 2019). Recent work by (Huber, Palmé, & Chao, 2023) and (Tian, Chao, Kulkarni, Goebel, & Fink, 2022) train neural network surrogates from aeroengine performance models to estimate component parameters. However, training such surrogates requires simulating the engine response across a large parameter space, and any change to the underlying physics necessitates new simulations and retraining. Moreover, ensuring that the surrogate remains consistent with the physical model requires extensive validation, increasing implementation complexity.

An alternative is to make the physics-based model itself differentiable. Inspired by recent progress in fields such as the Geosciences, where differentiable physics models have been developed using automatic differentiation libraries (Shen et al., 2023), this approach would allow the flexible integration of new physics without retraining the engine surrogate. To the best of our knowledge, no open-source differentiable aeroengine performance model currently exists. To address this

gap, we formulate the aeroengine performance model entirely using differentiable tensor computations compatible with automatic differentiation. In this way, gradients of any output, with respect to the model parameters, can be computed efficiently via backpropagation (Baydin, Pearlmutter, Radul, & Siskind, 2018). Consequently, the model can be seamlessly integrated with data-driven approaches, providing a powerful foundation for physics-informed machine learning in aeroengine condition monitoring (Karniadakis et al., 2021).

Specifically, this research introduces TurboGrad, an open-source differentiable aeroengine performance model for condition monitoring. The model is implemented in PyTorch and follows the same modeling principles of the original Gas turbine Simulation Program (GSPy) (W. P. J. Visser & Kogenhop, 2025). The thermodynamic cycle is expressed entirely through differentiable tensor operations, enabling gradient computation via backpropagation. The implementation is verified against GSPy for a single-spool turbojet configuration. A parameter estimation case demonstrates that gradient descent through backpropagation recovers ground-truth efficiency parameter changes from a nominal initialization.

This paper is divided as follows: Section 2 presents background literature used in this research. Next, Section 3 discusses details about the simulation, the problem formulation, and the case study. Then, Section 4 presents the results for the verification of our differentiable model with GSPy and the results of the case study. We discuss our work and future work in Section 5, followed by the conclusion in Section 6.

2. BACKGROUND

This section provides the necessary background information for the proposed framework. Section 2.1 introduces the component based modeling approach used for simulating aeroengine performance. Section 2.2 reviews automatic differentiation, which allows gradient-based optimization of the performance model.

2.1. Modeling Aeroengine Performance

The aeroengine performance model employed in this work follows the component based modeling principles of Gas turbine Simulation Program (GSP), originally developed at the Netherlands Aerospace Centre (NLR) and documented extensively by (W. Visser, 2015). GSP has been used for several decades for steady state and transient simulation of a wide variety of aeroengine types (Verbist, 2017). More recently, an open-source Python implementation of this framework, referred to as GSPy, has been released (W. P. J. Visser & Kogenhop, 2025).

Our research considers a single-spool turbojet performance model that computes sensor values such as pressure and temperatures at different stations along the engine gas path. The

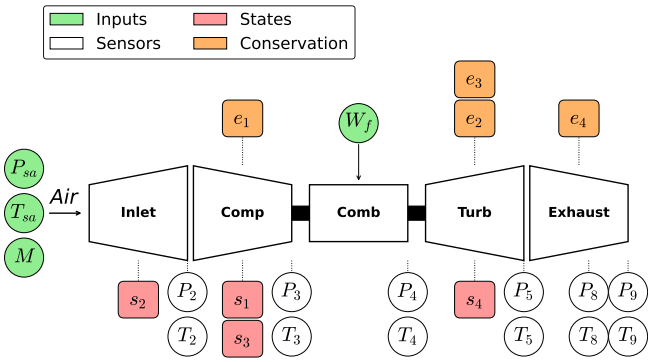

Figure 1. Turbojet performance model.

topology of such a zero-dimensional performance model is illustrated in Figure 1. The model consists of several component sub-models that relate to specific aero-thermodynamic processes in the engine. The flexibility of GSP is in its object oriented design, allowing for the development of new component models and extensions.

Simulating aeroengine system performance at a specific operating condition requires simulating the component sub-models in sequence, while enforcing consistency across the entire system:

$$\begin{aligned} \text{Inlet} \rightarrow \text{Compressor} \rightarrow \text{Combustor} \\ \text{Turbine} \rightarrow \text{Exhaust Nozzle} \end{aligned} \tag{1}$$

Each component receives the thermodynamic state of its predecessor, applies its transformation to the gas, and passes the result to the next component. Some components, such as the compressor and turbine, are attached to the same shaft and, thus, share a coupled state s. Because of this mechanical and thermodynamic coupling between the components, the solution cannot be obtained through a single forward evaluation. The coupling between components requires an iterative procedure to converge to a consistent operating point (Arias Chao, Kulkarni, Goebel, & Fink, 2022). We denote this system as $F(w, \theta)$, where w represents the operating conditions and θ the parameters of the model.

A valid operating point for the system should satisfy all conservation laws. These laws are embedded into the model as error equations e. For example, the power balance requires

$$e_3 = P_{comp} - \eta_{mech} \cdot P_{turb}, \tag{2}$$

where P_{comp} is the power required to drive the compressor, P_{turb} the power delivered by the turbine, and η_{mech} the mechanical efficiency. The objective is to reduce e_3 to zero, so that the power extracted from the turbine and used by the compressor are equal. Second-order methods such as

Newton-Raphson iteration can be used to adjust the engine states s and solve the system numerically (W. Visser, 2015).

2.2. Automatic Differentiation and Backpropagation

Fitting the parameters θ of a model requires computing the gradient of a scalar loss $\mathcal{L}$ with respect to each θ_i. For models with many parameters, approximating these gradients through finite differences (i.e. , numerical approximation) is impractical because each partial derivative requires an additional forward evaluation. Therefore, the total cost increases as $\mathcal{O}(n)$ in the number of parameters n, which is limiting when n is large (Baydin et al., 2018).

Automatic differentiation (AD), in turn, is a family of techniques that resolves this by computing exact derivatives of any function expressed as a computer program, at a cost comparable to a small constant multiple of one forward evaluation. The key insight is that any numerical computation can be decomposed into a finite sequence of elementary operations whose derivatives are known analytically from calculus (Griewank & Walther, 2008). Combining these local derivatives through the chain rule then yields the derivative of the overall computation.

AD can proceed in two directions. The forward mode propagates derivatives from inputs to outputs in a single pass and is efficient when the number of inputs is small relative to the number of outputs. In contrast, the reverse mode propagates derivatives from outputs back to inputs and is efficient when there are many inputs and a scalar output. This is exactly the setting of parameter optimization, where models tend to have a large number of parameters and, often, a single optimization metric.

To illustrate reverse mode concretely, consider the following function, whose algebraic structure mirrors the compressor outlet temperature relation discussed in Section 3.2:

$$f(x) = a \cdot \left(1 + \frac{b}{x}\right) \tag{3}$$

where a, b are constants and x is the trainable parameter. The forward pass decomposes this into a sequence of elementary operations by introducing intermediate variables u_i:

$$\begin{aligned} u_1 &= \frac{b}{x} \\ u_2 &= u_1 + 1 \\ u_3 &= u_2 \cdot a \end{aligned} \tag{4}$$

This sequence forms the computation graph of f, illustrated in Figure 2. Each node is an intermediate variable and each edge encodes a data dependency. In the reverse pass, the gradient of u_3 with respect to x is obtained by multiplying local derivatives backwards through the graph via the chain rule:

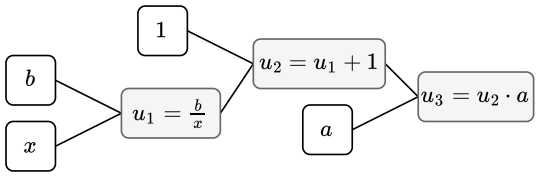

Figure 2. Computation graph for $f(x) = a \cdot (1 + b/x)$. Each node represents an intermediate variable; edges encode data dependencies. Constants a, b, and value 1 enter as leaf nodes.

$$\frac{du_3}{dx} = \frac{\partial u_3}{\partial u_2} \cdot \frac{\partial u_2}{\partial u_1} \cdot \frac{\partial u_1}{\partial x} \tag{5}$$

Each factor is a local derivative evaluated at the corresponding node during the forward pass:

$$\frac{\partial u_3}{\partial u_2} = a, \qquad \frac{\partial u_2}{\partial u_1} = 1, \qquad \frac{\partial u_1}{\partial x} = -\frac{b}{x^2} \tag{6}$$

Substituting yields the full derivative in a single backward sweep:

$$\frac{du_3}{dx} = a \cdot 1 \cdot \left(-\frac{b}{x^2}\right) = -\frac{ab}{x^2} \tag{7}$$

A single backward pass over the recorded computation graph yields the derivative with respect to x. While this example has only one trainable parameter, the same backward pass over a larger computation graph simultaneously yields the gradient with respect to all parameters at a cost comparable to one forward evaluation. This extends to any computation expressible as a graph of differentiable operations. The full gradient of a loss with respect to all parameters can be obtained via a single backward pass, regardless of whether the underlying components are neural network layers or physical equations. This is precisely the property TurboGrad exploits for modeling aeroengine performance.

3. METHODOLOGY

Our aeroengine performance model, TurboGrad, is implemented in PyTorch (Paszke et al., 2017), a widely used machine learning library with support for reverse-mode automatic differentiation. Figure 3 provides an overview with the relevant subsections for model development and the optimization process used for demonstration.

Section 3.1 formulates the optimization problem of fitting the model to sensor data. Our models follows a similar object-oriented approach of GSPy (W. P. J. Visser & Kogenhop, 2025); the main difference is the thermodynamics, which we implement in PyTorch for compatibility with automatic differentiation (Section 3.2). Section 3.3 presents the neural network surrogates for the compressor and turbine performance characteristics, and Section 3.4 describes the optimization process.

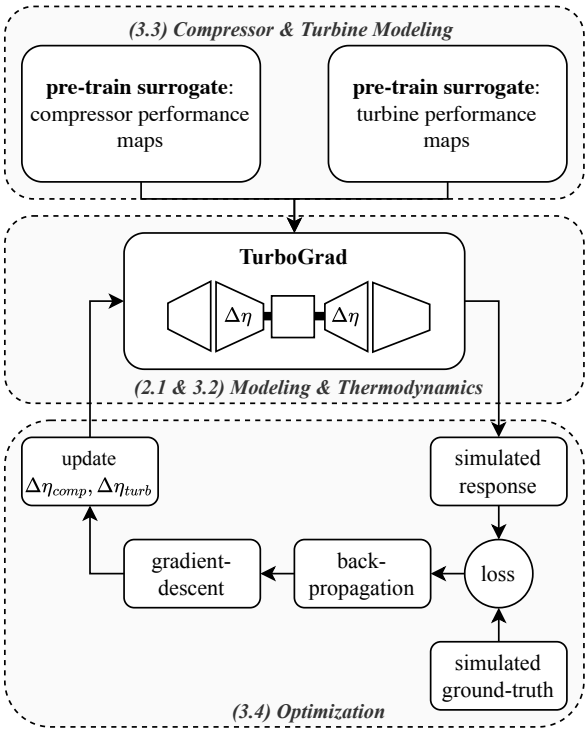

Figure 3. Overview of the methodology. Subsection labels indicate where each part is described.

3.1. Problem formulation

In this section, we introduce a steady-state engine performance model, which is evaluated at off-design operating conditions. The model defines an operating point w_i as

$$w_i = \begin{bmatrix} T_{\mathrm{sa}} & P_{\mathrm{sa}} & \mathrm{Ma} & W_f \end{bmatrix}^\top \tag{8}$$

where T_{sa} is the ambient static temperature, P_{sa} the ambient static pressure, Ma the Mach number, and W_f the fuel flow. The latter serves as a control input to the model.

Let θ denote the set of model parameters to be optimized and let F represent the steady-state aeroengine performance model. The simulated sensor values, $\hat{y}_i$, along the engine's gas path. The state-space representation of the system may be defined as follows:

$$0 = E(s_i, w_i, \theta) \tag{9}$$

$$\hat{y}_i = F(s_i^\star, w_i, \theta), \tag{10}$$

where E represents the system conservation errors and F the predicted engine response for an operating condition i. Evaluating the engine response requires finding a solution for the system state $s^\star$. The Newton-Raphson method is used to find the root of the system within a specified tolerance ε (W. Visser, 2015):

$$\|E(s_i^\star, w_i; \theta)\|_2 \leq \varepsilon, \qquad \varepsilon > 0, \tag{11}$$

where ε is the solver tolerance and $\|_2$ is the L2 norm of the error equations. Since E is a vector of conservation error equations, it may be denoted as follows:

$$E = \begin{bmatrix} e_1(s_i, w_i, \theta) \\ \vdots \\ e_n(s_i, w_i, \theta) \end{bmatrix} \in \mathbb{R}^n. \tag{12}$$

Here, each conservation error e enforces a physical constraint such as mass-flow continuity (conservation of mass) or shaft power balance (conservation of energy) to find a valid operating point. For the turbojet engine considered in this work, s contains 4 states: the normalized shaft speed, corrected mass flow, and two operating-line parameters which indicate the performance characteristic of the compressor and turbine. The system is square because there are also 4 conservation equations.

Given a target dataset $\mathcal{D} = \{(w_i, y_i)\}_{i=1}^{N_{\mathrm{data}}}$ with operating points and sensor measurements y_i, the parameters of the model can be optimized to fit the data by minimizing

$$\min_\theta \ \mathcal{L}(\theta) = \frac{1}{N_{\mathrm{data}}} \sum_{i=1}^{N_{\mathrm{data}}} \sum_{k \in \mathcal{K}} \ell(\hat{y}_{i,k}(\theta),\, y_{i,k})\,. \tag{13}$$

Here, $\mathcal{K}$ represents the sensors $[T_3, P_3, T_4, P_4, T_5, P_5, N]$. A mean squared relative error is used as loss function:

$$\ell(\hat{y}_{i,k},\, y_{i,k}) = \left(\frac{\hat{y}_{i,k} - y_{i,k}}{|y_{i,k}|} \right)^2, \tag{14}$$

which balances sensor contributions with different physical units and magnitudes. Relative error formulations are also consistent with common practice in gas path analysis, where deviations from a baseline are typically interpreted in percentages as discussed in (Volponi & Tang, 2025).

3.2. Thermodynamics

The reference performance model, GSPy, utilizes the Cantera[1] library for modeling the gas thermodynamics. However, this library is not compatible with automatic differentiation. Instead, to enable gradient flow through the thermodynamic engine cycle, we implemented a thermodynamic model inspired by the Reactorch package (Ji & Deng, 2020). The gas is treated as a mixture of chemical species whose composition changes through combustion. Following the same approach as Cantera, the thermodynamic properties of each species, such as c_p, h, and s, are evaluated using NASA-7 polynomi-

[1] https://cantera.org/

als (McBride, 2002) as polynomial functions of temperature; bulk gas properties then follow from mass fraction weighted averaging over all species in the mixture. All operations were implemented as differentiable tensor computations, ensuring compatibility with automatic differentiation.

A recurring procedure in the simulation is finding the outlet temperature consistent with a thermodynamic constraint. While a constant specific heat ratio γ simplifies this to a closed form (i.e $T_3 = T_2\big(1 + (\pi_c^{(\gamma-1)/\gamma} - 1)/\eta_{\text{comp}}\big)$), γ is itself a function of temperature and gas composition. The outlet temperature is therefore found via scalar Newton iteration. The gas state solvers are embedded within the component forward passes and also remain differentiable.

3.3. Compressor & Turbine Modeling

The behavior of the compressor and turbine is characterized by component performance maps. The maps encode the relationships between corrected speed N_c, corrected mass flow W_c, pressure ratio PR, and isentropic efficiency η over the admissible operating region, as illustrated in Figure 4. These quantities govern the component's aerothermodynamic performance throughout the operating region. The maps used in this study are taken from GSPy, which stores the aforementioned quantities as discrete data points.

The map coordinates are queried during simulation after interpolation. However, interpolation methods from standard libraries such as SciPy are typically not compatible with automatic differentiation. To resolve this, the interpolator is replaced by a neural network surrogate (Ghorbanian & Gholamrezaei, 2007). The surrogate is a multilayer perceptron trained on samples generated from the interpolator, learning the mapping $(N_c, \beta) \to (W_c, PR, \eta)$, where β is an auxiliary coordinate that parametrizes the operating point on the map (Kurzke, 1996). The architecture is summarized in Table 1.

Because the surrogate is implemented in PyTorch, map evaluations are fully differentiable. We note that the use of a neural network surrogate is not a requirement of the framework; any differentiable map representation can be used. It must be noted, however, that approximation errors in the surrogate maps may still propagate through the performance model and affect the convergence behavior described in Eq. 11. To quantify this effect, the verification procedure in Section 4.1 compares TurboGrad against GSPy using both the original interpolated maps and the neural network surrogates.

To represent performance deterioration, adaptation factors were applied to the map outputs. For efficiency, the adapted value is:

$$\eta_{\text{adp}} = (1 + \Delta\eta)\ \eta_{\text{map}}, \tag{15}$$

where a negative $\Delta\eta$ shifts the efficiency curve downward across the entire operating range. For instance, $\Delta\eta = -0.04$ corresponds to a 4% efficiency reduction, as visualized in

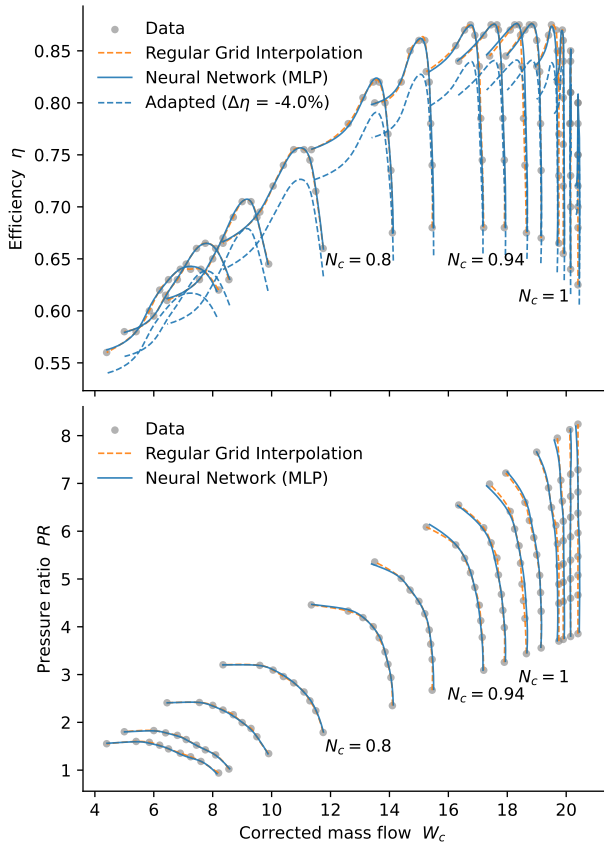

Figure 4. Compressor performance map used in this study. Markers denote discrete map data. Curves illustrate (i) interpolation, (ii) the neural network surrogate, and (iii) an efficiency adaptation shifting the map.

Figure 4. Such modification factors are standard practice in adaptive simulation and performance map tuning (Stamatis, Mathioudakis, & Papailiou, 1990). Therefore, in this work, the optimizable model parameters are the efficiency adaptation factors for both turbomachinery components:

$$\theta = \{\Delta\eta_{\text{comp}}, \Delta\eta_{\text{turb}}\}, \tag{16}$$

where $\Delta\eta_{\text{comp}}$ and $\Delta\eta_{\text{turb}}$ are the compressor and turbine efficiency adaptation parameters, respectively.

3.4. Optimization

Optimization of the aeroengine performance model parameters θ follows a two-step procedure, repeated over multiple epochs. At each epoch, the engine first simulates the sensor values at every operating point in the dataset by solving the system of equations as denoted by Eq. (11). The predicted sensor outputs are then compared to the targets, and the model parameters are updated by gradient descent. The procedure is summarized in Algorithm 1.

Table 1. Neural network surrogate hyperparameters for component maps.

Parameter	Value
Hidden layers	3
Neurons per layer	128
Activation	GELU
Batch size	256

3.4.1. Inner loop: root finding

A solution for the aeroengine performance model is found using the Krylov-based Newton solver. The solver iterates from an initial guess until the convergence criterion from Eq. (11) is satisfied, with tolerance $\varepsilon = 10^{-6}$ and a maximum of 20 iterations. A solution for each operating point is first computed before optimizing the model parameters with gradient-descent.

3.4.2. Outer loop: gradient descent

Once all operating points have been simulated, the gradients $\nabla_\theta \mathcal{L}$ are computed via backpropagation through the post-solve evaluation of F, as described in Section 3.1. The parameters are then updated using gradient descent with a learning rate of $\alpha = 0.2$. Training proceeds until the loss falls below $\varepsilon_\mathcal{L}$ or the maximum number of epochs $T_{\max} = 120$ is reached. The parameters are initialized at $\theta^{(0)} = (1, 1)$, corresponding to no adaptation (i.e., nominal performance).

Algorithm 1 Estimating aeroengine performance model parameters via gradient descent.

Require: Dataset $\mathcal{D} = \{(w_i, y_i)\}_{i=1}^{N_{\text{data}}}$, initial parameters $\theta^{(0)}$, learning rate α, solver tolerance ε, maximum epochs $T_{\max}$, loss tolerance $\varepsilon_\mathcal{L}$
Ensure: Optimized parameters θ

$\theta \leftarrow \theta^{(0)}$
for $t = 1, \ldots, T_{\max}$ **do** ▷ Outer Loop
  for $i = 1, \ldots, N_{\text{data}}$ **do** ▷ Inner Loop
    Solve $s_i^\star$ such that $\|E(s_i^\star, w_i, \theta)\|_2 \leq \varepsilon$
    $\hat{y}_i \leftarrow F(w_i, \theta)$ ▷ sensor predictions
  end for
  $\mathcal{L}(\theta) \leftarrow \frac{1}{N_{\text{data}}} \sum_{i=1}^{N_{\text{data}}} \sum_{k \in \mathcal{K}} \ell(\hat{y}_{i,k}(\theta), y_{i,k})$
  if $\mathcal{L}(\theta) \leq \varepsilon_\mathcal{L}$ **then**
    break ▷ early stopping
  end if
  $\nabla_\theta \mathcal{L} \leftarrow \texttt{backprop}(\mathcal{L}, \theta)$
  $\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}$
end for
return θ

4. RESULTS

This section presents two sets of results. First, Section 4.1 verifies the differentiable TurboGrad model against GSPy. Second, Section 4.2 demonstrates the gradient-based procedure on a parameter estimation problem.

4.1. Verification

To verify that TurboGrad reproduces the behavior of GSPy, both models are evaluated on a common set of 112 off-design operating points spanning 7 ambient conditions and 16 fuel flow input values. The ambient conditions cover a range of temperatures, pressures, and Mach numbers representative of ground-level to high-altitude flight, with fuel flow ranging from 0.25 to 0.40 kg/s. Since TurboGrad replaces the interpolated component maps with neural network surrogates, we also compare what the results would have been when using the interpolated maps from GSPy to isolate the thermodynamic implementation from Section 3.2.

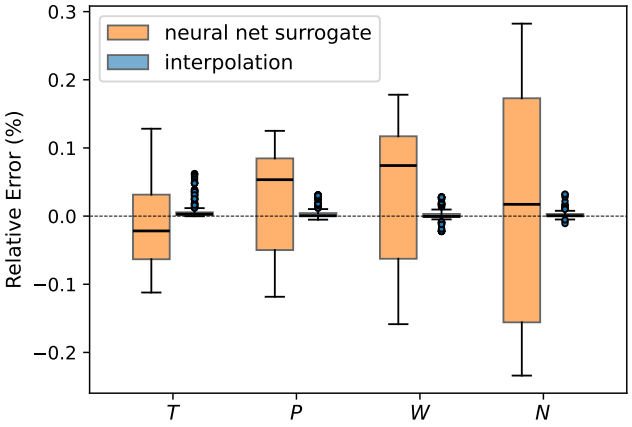

Figure 5. Relative percentage error between TurboGrad and GSPy for the temperature T, pressure P, mass flow W, and spool speed N.

Figure 5 shows the relative error of the successful simulations for gas-path quantities at different stations along the engine gas-path: temperature T, pressure P, mass flow W, and shaft speed N. Not all operating points converge under the solver tolerance for TurboGrad: 98 of 112 points converge with the interpolated maps and 102 of 112 with the neural network surrogate. Non-converged points are excluded from the comparison, as their residual errors can produce relative deviations that are an order of magnitude larger than those of converged solutions. With the interpolated maps, relative errors remain below 0.1% across all quantities, confirming that the PyTorch reimplementation of the thermodynamic cycle and solver is numerically consistent with GSPy. With the neural network surrogate, the errors increase but remain within ±0.3%. Hence, the additional error is attributed to the approximation introduced by the surrogate, not to the engine model itself.

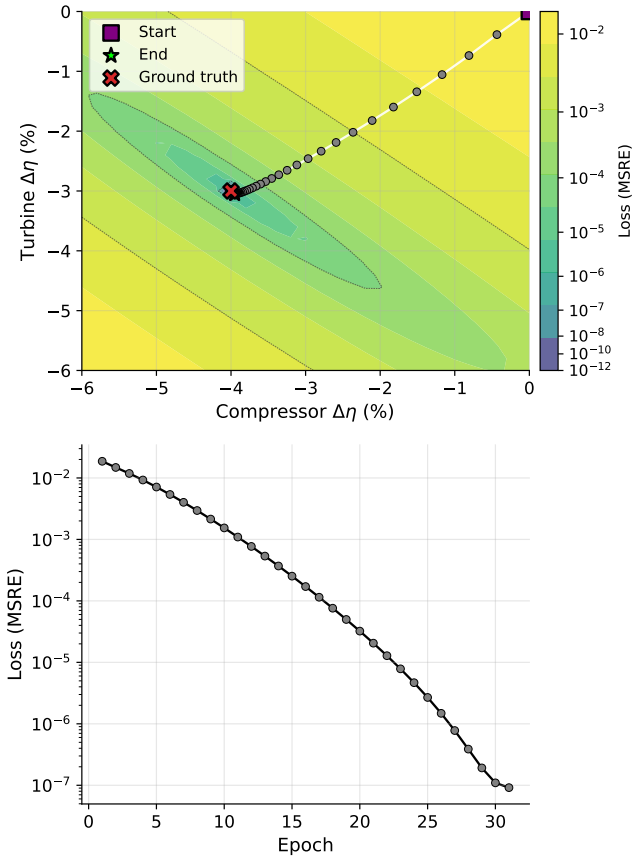

Figure 6. Top: optimization trajectory on the loss surface, starting from the nominal point and converging near the ground truth. Bottom: loss (MSRE) versus epoch.

4.2. Parameter Estimation

To demonstrate gradient-based optimization, we consider a synthetic parameter estimation problem. The ground-truth model is configured with efficiency adaptations of $\Delta\eta_{\text{comp}} = -4\%$ and $\Delta\eta_{\text{turb}} = -3\%$, simulating a degraded compressor and turbine. Target sensor measurements y_i are generated by simulating this degraded conditon over 22 operating points, producing values for $[T_3, P_3, T_4, P_4, T_5, P_5, N]$. The optimization starts from the nominal initialization $\theta^{(0)} = (1, 1)$ and attempts to recover the imposed adaptations from the sensor data with gradient descent.

Figure 6 illustrates the convergence behavior of the gradient-based optimization. The top panel displays the optimization trajectory on the two-dimensional loss landscape. Starting from the nominal point (top-right corner), gradient descent follows a direct path toward the ground-truth parameters, converging in approximately 30 epochs. The bottom panel shows the corresponding loss curve, which decreases by over five orders of magnitude from $\sim 10^{-2}$ to below 10^{-7}. The final parameter estimates are $\Delta\hat{\eta}_{\text{comp}} = -3.95\%$ and $\Delta\hat{\eta}_{\text{turb}} = -3.05\%$, closely matching the ground truth.

5. DISCUSSION AND FUTURE WORK

A key outcome of this work is the demonstration that aero-engine performance models can be reformulated as fully differentiable systems without sacrificing numerical consistency. The verification results show that reformulating the thermodynamic cycle in PyTorch matches closely with expected values. With the original interpolation-based maps, relative errors remain below 0.1% across all gas-path quantities, confirming strong numerical agreement with GSPy.

TurboGrad's modular architecture allows gradients to backpropagate through any differentiable component. This is particularly useful for end-to-end optimization and facilitates the integration of new physical components or data-driven models without requiring a surrogate to be retrained.

The case study presented in this work considers an idealized parameter estimation scenario using synthetic, noise-free data generated by the model itself. In practice, however, real-world sensor measurements are subject to noise, bias, and model mismatch, as the performance model is only an approximation of the true physical system. Although these issues are not addressed in this work, our model's support for automatic differentiation enables tighter integration with deep learning architectures. Such hybrid combinations are widely hypothesized to be more robust when applied to real-world data.

Future work should focus on validating the model using real sensor data from modern aircraft engines. Therefore, a natural first direction is to extend the current application to multi-spool turbofan engines. In addition, incorporating neural networks into the training pipeline offers a promising direction, where fleet-level patterns could be learned and transferred to improve predictions for individual engines through transfer learning. Vectorizing the simulation to evaluate multiple operating points simultaneously would substantially reduce computation time and facilitate online condition monitoring.

6. CONCLUSION

This paper presents TurboGrad, an open-source differentiable aeroengine performance model implemented in PyTorch. The model reformulates the component-based modeling approach of GSPy so that the entire gas-path computation is compatible with automatic differentiation. Verification against GSPy confirmed strong numerical consistency, and a synthetic parameter estimation case demonstrated that gradients backpropagate successfully through the thermodynamic cycle. By providing an open and differentiable performance model, this study lowers the barrier to research on hybrid architectures that combine physics-based and data-driven models within a single optimization framework. Future work will focus on validation with real engine data, extension to more complex engine architectures (e.g., multi-spool turbofans), and the in-

tegration of data-driven components to support fleet-level modeling and transfer learning.

CODE AVAILABILITY

The proposed model, TurboGrad, is publicly accessible at: https://github.com/daniel-cisneros-acevedo/turbograd under the MIT license.

ACKNOWLEDGMENT

This work was supported by the BrightSky project, funded by the R&D Mobiliteitsfonds from the Netherlands Enterprise Agency (RVO) and commissioned by the Ministry of Economic Affairs and Climate Policy.

REFERENCES

Arias Chao, M., Kulkarni, C., Goebel, K., & Fink, O. (2022, January). Fusing physics-based and deep learning models for prognostics. *Reliability Engineering & System Safety*, *217*, 107961.

Baydin, A. G., Pearlmutter, B. A., Radul, A. A., & Siskind, J. M. (2018, February). *Automatic differentiation in machine learning: a survey.* arXiv. (arXiv:1502.05767 [cs])

Borguet, S. (2012, June). Variations on the Kalman filter for enhanced performance monitoring of gas turbine engines.

Chao, M. A., Lilley, D. S., Mathé, P., & Schloßhauer, V. (2015, August). Calibration and Uncertainty Quantification of Gas Turbine Performance Models. American Society of Mechanical Engineers Digital Collection.

Fentaye, A. D., Baheta, A. T., Gilani, S. I., & Kyprianidis, K. G. (2019, July). A Review on Gas Turbine Gas-Path Diagnostics: State-of-the-Art Methods, Challenges and Opportunities. *Aerospace*, *6*(7), 83. (Number: 7)

Ghorbanian, K., & Gholamrezaei, M. (2007, January). Axial Compressor Performance Map Prediction Using Artificial Neural Network. In *Volume 6: Turbo Expo 2007, Parts A and B* (pp. 1199–1208). Montreal, Canada: ASMEDC.

Griewank, A., & Walther, A. (2008). *Evaluating derivatives: principles and techniques of algorithmic differentiation* (2. ed ed.). Philadelphia: SIAM.

Huber, L. G., Palmé, T., & Chao, M. A. (2023, June). Physics-Informed Machine Learning for Predictive Maintenance: Applied Use-Cases. In *2023 10th IEEE Swiss Conference on Data Science (SDS)* (pp. 66–72).

Ji, W., & Deng, S. (2020). *ReacTorch: A differentiable reacting flow simulation package in PyTorch.*

Karniadakis, G. E., Kevrekidis, I. G., Lu, L., Perdikaris, P., Wang, S., & Yang, L. (2021, June). Physics-informed machine learning. *Nature Reviews Physics*, *3*(6), 422–440.

Kurzke, J. (1996, June). How to Get Component Maps for Aircraft Gas Turbine Performance Calculations. In *Volume 5: Manufacturing Materials and Metallurgy; Ceramics; Structures and Dynamics; Controls, Diagnostics and Instrumentation; Education; General* (p. V005T16A001). Birmingham, UK: American Society of Mechanical Engineers.

Li, Y. G., Ghafir, M. F. A., Wang, L., Singh, R., Huang, K., & Feng, X. (2011, March). Nonlinear Multiple Points Gas Turbine Off-Design Performance Adaptation Using a Genetic Algorithm. *Journal of Engineering for Gas Turbines and Power*, *133*(071701).

McBride, B. J. (2002). *NASA Glenn Coefficients for Calculating Thermodynamic Properties of Individual Species*. National Aeronautics and Space Administration, John H. Glenn Research Center at Lewis Field. (Google-Books-ID: TAEVAQAAIAAJ)

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., … Desmaison, A. (2017). Automatic differentiation in PyTorch.

Shen, C., Appling, A. P., Gentine, P., Bandai, T., Gupta, H., Tartakovsky, A., … Lawson, K. (2023, July). Differentiable modeling to unify machine learning and physical models and advance Geosciences. *Nature Reviews Earth & Environment*, *4*(8), 552–567. (arXiv:2301.04027 [cs])

Stamatis, A., Mathioudakis, K., & Papailiou, K. D. (1990, April). Adaptive Simulation of Gas Turbine Performance. *Journal of Engineering for Gas Turbines and Power*, *112*(2), 168–175.

Tian, Y., Chao, M. A., Kulkarni, C., Goebel, K., & Fink, O. (2022, February). Real-time model calibration with deep reinforcement learning. *Mechanical Systems and Signal Processing*, *165*, 108284.

Verbist, M. L. (2017). *Gas Path Analysis for Enhanced Aero-Engine Condition Monitoring and Maintenance* (PhD Thesis). Delft University of Technology.

Visser, W. (2015). *Generic analysis methods for gas turbine engine performance: The development of the gas turbine simulation program GSP* (Dissertation (TU Delft)).

Visser, W. P. J., & Kogenhop, O. (2025). *GSPy: Python propulsion and power system performance modelling and simulation tool.*

Volponi, A. J., & Tang, L. (2025). *Principles of Gas Path Analysis* (1st ed.). Boca Raton: CRC Press.

Vu, D. Q., Razakarivony, S., Marnissi, Y., & Nocture, M. (2024, August). A Comprehensive Literature Review on the Resolution of Turbine Engine Performances' Inverse Problems. American Society of Mechanical Engineers Digital Collection.