TENG: Time-Evolving Natural Gradient for Solving PDEs With Deep Neural Nets Toward Machine Precision

Zhuo Chen [1 2 3] **Jacob McCarran** [1 2 *] **Esteban Vizcaino** [1 2 *] **Marin Soljačić** [1 2 3] **Di Luo** [1 2 3 4]

Abstract

Partial differential equations (PDEs) are instrumental for modeling dynamical systems in science and engineering. The advent of neural networks has initiated a significant shift in tackling these complexities though challenges in accuracy persist, especially for initial value problems. In this paper, we introduce the *Time-Evolving Natural Gradient (TENG)*, generalizing time-dependent variational principles and optimization-based time integration, leveraging natural gradient optimization to obtain high accuracy in neural-network-based PDE solutions. Our comprehensive development includes algorithms like TENG-Euler and its high-order variants, such as TENG-Heun, tailored for enhanced precision and efficiency. TENG's effectiveness is further validated through its performance, surpassing current leading methods and achieving *machine precision* in step-by-step optimizations across a spectrum of PDEs, including the heat equation, Allen-Cahn equation, and Burgers' equation.

1. Introduction

Partial differential equations (PDEs) hold profound significance in both the theoretical and practical realms of mathematics, science, and engineering. They are essential tools for describing and understanding a multitude of phenomena that exhibit variations across different dimensions and points in time. The study and solution of PDEs have driven advancements in numerical analysis and computational methods, as many real-world problems modeled by PDEs are too complex for analytical solutions. The long-pursued quest for an efficient and accurate numerical PDE solver continues to be a central endeavor passionately pursued by research communities.

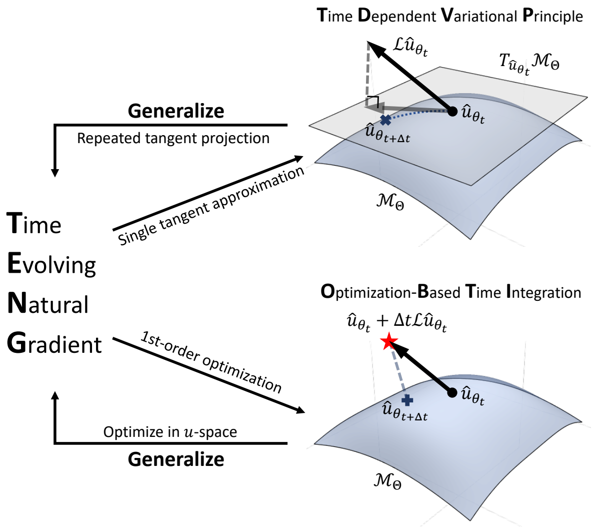
Time Dependent Variational Principle
$\mathcal{L}\hat{u}_{\theta_t}$
$T_{\hat{u}_{\theta_t}}\mathcal{M}_\Theta$
$\hat{u}_{\theta_t}$
$\hat{u}_{\theta_{t+\Delta t}}$
$\mathcal{M}_\Theta$
Generalize
Repeated tangent projection
Single tangent approximation
Time
Evolving
Natural
Gradient
1st-order optimization
Optimize in u-space
Generalize
Optimization-Based Time Integration
$\hat{u}_{\theta_t} + \Delta t\mathcal{L}\hat{u}_{\theta_t}$
$\hat{u}_{\theta_t}$
$\hat{u}_{\theta_{t+\Delta t}}$
$\mathcal{M}_\Theta$

Figure 1. TENG generalizes the existing TDVP and OBTI methods. Within a single time step, TDVP projects the update direction $\mathcal{L}\hat{u}_{\theta_t}$ onto the tangent space of the neural network manifold $T_{\hat{u}_{\theta_t}}\mathcal{M}_\Theta$ at time t, and evolves the parameters θ according to this tangent space projection. OBTI optimizes θ to obtain an approximation to the target function $\hat{u}_{\theta_t} + \Delta t\mathcal{L}\hat{u}_{\theta_t}$ on the manifold $\mathcal{M}_\Theta$. Generalizing these two methods, TENG defines the loss function directly in the u-space and optimizes the loss function via repeated projections to the tangent space $T_{\hat{u}_{\theta_t}}\mathcal{M}_\Theta$.

In recent years, the introduction of machine learning (ML) into the study of PDEs (Han et al., 2017; Yu et al., 2018; Long et al., 2018; Carleo & Troyer, 2017; Raissi et al., 2019; Li et al., 2020; Lu et al., 2019; Han et al., 2018; Sirignano & Spiliopoulos, 2018; Chen et al., 2022; 2023b) has marked a transformative shift in both fields, particularly highlighted in the realms of computational mathematics and data-driven discovery. Machine learning offers new possibilities for tackling the complexities inherent in PDEs, which often pose significant challenges for traditional numerical methods due to high dimensionality, nonlinearity, or chaotic behavior. By leveraging neural networks' ability

[1]{chenzhuo,mccarran,edviz,soljacic,diluo}@mit.edu. [2]Department of Physics, Massachusetts Institute of Technology. [3]NSF AI Institute for Artificial Intelligence and Fundamental Interactions. [4]Department of Physics, Harvard University. [*]Equal contribution.
Corresponding author: Di Luo.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

to approximate complex functions, algorithms have been developed to solve, simulate, and even discover PDEs from data, circumventing the need for explicit formulations.

Partial differential equations with initial value problems, crucial in describing the evolution of dynamical systems, represent a fundamental class within the realm of PDEs. Despite the promising advancements made by machine learning techniques in approximating the solutions to these complex PDEs, they frequently encounter difficulties in maintaining high levels of accuracy, a challenge that becomes particularly pronounced when navigating the intricate initial conditions. This challenge largely originates from the cumulative and propagative of errors in PDE solvers over time, necessitating precise solutions at each time step for accuracy. Although various training strategies, both global-in-time training (Müller & Zeinhofer, 2023) and sequential-in-time training (Chen et al., 2023a; Berman & Peherstorfer, 2023), have been proposed to address this issue, it continues to stand as a critical challenge in the field.

Contributions. In this paper, we introduce a highly accurate and efficient approach for tackling the above challenge by introducing **Time-Evolving Natural Gradient (TENG)**. Our key contributions are three-fold and highlighted as follows:

- Propose the TENG method which generalizes two fundamental approaches in the field, time-dependent variational principle (TDVP) and optimization-based time integration (OBTI), and achieves highly accurate results by integrating natural gradient with sequential-in-time optimization.
- Develop efficient algorithms with sparse update for the realization of TENG, including the basic TENG-Euler and the highly accurate higher-order versions, such as TENG-Heun.
- Demonstrate that our approach obtains orders of magnitude better performances than state-of-the-art methods such as OBTI, TDVP with sparse updates, and PINN with energy natural gradient, and achieves machine precision accuracy during per-step optimization on a variety of PDEs, including the heat equation, Allen-Cahn equation, and Burgers' equation.

2. Related work

Machine Learning in PDEs. Machine learning has been used to solve PDEs by using neural networks as function approximators to the solutions. In general, there are two types of strategies, global-in-time optimization and sequential-in-time optimization. Global-in-time optimization includes the physics-informed neural network (PINN) (Raissi et al., 2019; Wang & Perdikaris, 2023; Wang et al., 2021b; Sirignano & Spiliopoulos, 2018; Wang et al., 2021a), which optimizes the neural network representation over time and space simultaneously, or deep Ritz method (Weinan et al., 2021; Yu et al., 2018) when the variational form of the PDE exists. In contrast, sequential-in-time optimization (sometimes also called neural Galerkin method) only uses the neural network to represent the solution at a particular time step and updates the neural network representation step-by-step in time. There are different approaches to achieving such updates, including time-dependent variational principle (TDVP) (Dirac, 1930; Koch & Lubich, 2007; Carleo & Troyer, 2017; Du & Zaki, 2021; Berman & Peherstorfer, 2023) and optimization-based time integration (OBTI) (Chen et al., 2023a; Kochkov & Clark, 2018; Gutiérrez & Mendl, 2022; Luo et al., 2022; 2023; Sinibaldi et al., 2023). Machine Learning has also been applied to model PDEs based on data. Such data-driven approaches include neural ODE (Chen et al., 2018), graph neural network methods (Pfaff et al., 2020; Sanchez-Gonzalez et al., 2020), neural Fourier operator (Li et al., 2020), and DeepONet (Lu et al., 2019).

Natural Gradient. The concept of natural gradients, first introduced by Amari (Amari, 1998) has become a cornerstone in the evolution of optimization techniques within machine learning. These methods modify the update direction in gradient-based optimization as a second-order method, typically involving using the Fisher matrix. Distinct from traditional gradient methods due to its consideration of the underlying data geometry, natural gradient descent leads to faster and more effective convergence in various scenarios. Natural gradient descent and its variants have found widespread application in areas such as neural network optimization (Peters et al., 2003; Pascanu & Bengio, 2013; Zhang et al., 2019), reinforcement learning (Peters et al., 2003; Kakade, 2001), quantum optimization (Stokes et al., 2020), and PINN training (Müller & Zeinhofer, 2023).

3. Problem Formulation and Challenges

3.1. Problem formulation

Given a spatial domain $\mathcal{X} \subseteq \mathbb{R}^d$ and temporal domain $\mathcal{T} \subset \mathbb{R}$, let u be a function $\mathcal{X} \times \mathcal{T} \to \mathbb{R}$ that satisfies the following initial value problem of a PDE

$$\begin{aligned} &\frac{\partial u(x,t)}{\partial t} = \mathcal{L}u(x,t) \quad \text{for} \quad (x,t) \in \mathcal{X} \times \mathcal{T} \quad \text{and} \\ &u(x,0) = u_0(x), \end{aligned} \tag{1}$$

with appropriate boundary conditions. The sequential-in-time optimization approach uses neural network to parameterize the solution of the PDE at a particular time step $t \in \mathcal{T}$ as $\hat{u}_{\theta_t}(x) : \Theta \times \mathcal{X} \to \mathbb{R}$, where the parameters have an explicitly time dependence $\theta_t : \mathcal{T} \to \mathbb{R}^{N_p}$ (with N_p the number of parameters) and evolves over time. To solve the PDE, the neural network is first optimized to match the initial condition $\hat{u}_{\theta_0}(x) = u_0(x)$, and then optimized in a time-step-by-time-step fashion to update the parameters.

We contrasted this with the global-in-time optimization method, such as PINN (Raissi et al., 2019), where the neural network is used to parameterize the solution for all time $\hat{u}_\theta(x,t)$ with a single set of parameters. In this context, a loss function that gives rise to the global solution of the PDE is used to optimize the parameters.

3.2. Time Dependent Variational Principle

Time-dependent variational principle (TDVP) is an existing sequential-in-time method. It aims to derive an ODE in the parameter θ-space based on the function u-space PDE (Fig. 1). The most commonly used projection method is the Dirac–Frenkel variational principle (Dirac, 1930), which defines the ODE by solving the following least square problem at each time step

$$\partial_t\theta = \underset{\partial_t\theta\in\mathbb{R}^{N_p}}{\arg\min} \left\|\mathcal{L}\hat{u}_\theta(\cdot) - \textstyle\sum_j J_{(\cdot),j}\partial_t\theta_j\right\|^2_{L^2(\mathcal{X})}, \quad (2)$$

where $J_{(x),j} := \partial\hat{u}_\theta(x)/\partial\theta_j$ is the Jacobian.

Denoting the function space of u with $\mathcal{U}$, the manifold of neural network parameterized functions $\hat{u}_\theta$ with $\mathcal{M}_\Theta$, and the tangent space to the manifold at $\hat{u}_{\theta_t}$ with $T_{\hat{u}_{\theta_t}}\mathcal{M}_\Theta$, Eq. (2) gives the orthogonal projection of the evolution direction $\partial_t u = \mathcal{L}u$ onto the tangent space $T_{\hat{u}_{\theta_t}}\mathcal{M}_\Theta$ generated by the pushfoward of $\partial_t\theta$. The resulting ODE in the θ-space can then be evolved in discrete time steps using numerical integrators such as the 4th-order Runge–Kutta (RK4) method.

Limitations. The Dirac–Frenkel variational principle produces the orthogonal projection of the evolution onto the tangent space $T_{\hat{u}_{\theta_t}}\mathcal{M}_\Theta$ at $\hat{u}_{\theta_t}$ during each time step. For nonzero time step sizes Δt, however, the result becomes only an approximation to the optimal projection of the target solution onto the manifold $\mathcal{M}_\Theta$. The evolution on $\mathcal{M}_\Theta$ can also deviate from the projected direction on $T_{\hat{u}_{\theta_t}}\mathcal{M}_\Theta$ due to nonzero time step sizes, which gives rise to the following consequence: although Eq. (2) is reparameterization invariant, its nonzero Δt version is not (see Appendix Theorem A.1 for detail). In addition, the least square problem in Eq. (2) is often ill-conditioned and the solution could be sensitive to the number of parameters, the number of samples used, and the regularization method. Although Ref. (Berman & Peherstorfer, 2023) proposed a sparse update method, where a random subset of parameters are updated at each time step, it is still hard to verify whether such choice gives the best projection in practice. Meanwhile, the solution of Eq. (2) after regularization could be different from optimal.

3.3. Optimization-based Time Integration

Optimization-based time integration (OBTI) is an alternative sequential-in-time method. It directly discretizes the PDE into time steps in the original function u-space; in each time step, OBTI first finds the next-time-step target function u_{target} based on the current-time-step $\hat{u}_{\theta_t}$, and then optimizes the next-time-step parameters $\theta_{t+\Delta t}$ by minimizing a loss function

$$\theta_{t+\Delta t} = \underset{\theta\in\Theta}{\arg\min}\, L(\hat{u}_\theta, u_{\text{target}}). \quad (3)$$

Depending on the discrete-time integration schemes used, u_{target} can be different. The most commonly used integration scheme is the forward Euler's method, where $u_{\text{target}} = \hat{u}_{\theta_t} + \Delta t\mathcal{L}\hat{u}_{\theta_t}$ Some typical loss functions used in OBTI methods include the L^2-distance, the L^1-distance, and the KL-divergence.

Limitations. Although the optimal solution to Eq. (3) gives the best approximation of u_{target} in $\mathcal{M}_\Theta$, in practice, the optimization can be very difficult with a non-convex landscape. Common optimizers such as Adam and BFGS (L-BFGS) often require a significant number of iterations to obtain an acceptable loss value. Since this optimization has to be repeated over all time steps, the accumulation of error and cost often results in poor performance. In addition, the integration scheme used in current implementations of OBTI is often limited to the forward Euler's method, which requires small time steps and further amplifies the issue of error accumulation and cost. We note that while higher-order integration schemes have been explored in prior works, they either involve applying $\mathcal{L}$ multiple times on $\hat{u}_\theta$ (Donatella et al., 2023) or require differentiating through $\mathcal{L}\hat{u}_\theta$ with respect to θ (Luo et al., 2022; 2023), both of which requires high-order differentiation, leading to stability issues and further increase of the cost.

4. Time-Evolving Natural Gradient (TENG)

4.1. Generalization from TDVP and OBTI

We first make the following observation.

Observation: *TDVP can be viewed as solving Eq.* (3) *with the (squared) L^2-distance as the loss function using a single tangent space approximation at each time step.*

Proof. At time t, the neural network manifold $\mathcal{M}_\Theta$ can be approximated at the point $\hat{u}_{\theta_t}$ by its tangent space as

$$\hat{u}_{\theta+\delta\theta} = \hat{u}_\theta + \sum_j \frac{\partial\hat{u}_\theta}{\partial\theta_j}\delta\theta_j + \mathcal{O}(\delta\theta^2), \quad (4)$$

Let $L(\hat{u}_{\theta+\delta\theta}, u_{\text{target}}) = \|\hat{u}_{\theta+\delta\theta} - u_{\text{target}}\|^2_{L^2(\mathcal{X})}$. For small δt, $u_{\text{target}} = \hat{u}_\theta + \delta t\mathcal{L}\hat{u}_\theta + \mathcal{O}(\delta t^2)$. Keeping everything to first order, the loss function takes its minimum when

$$\delta\theta = \underset{\delta\theta\in\mathbb{R}^{N_p}}{\arg\min} \left\| \delta t \mathcal{L}\hat{u}_\theta(\cdot) - \sum_j \frac{\partial \hat{u}_\theta(\cdot)}{\partial \theta_j}\delta\theta_j \right\|^2_{L^2(\mathcal{X})}. \quad (5)$$

Dividing both sides by δt recovers the TDVP (Eq. (2)). □

Inspired by such observation, we introduce the time-evolving natural gradient (TENG) method, which generalizes TDVP and OBTI methods in the following way:

TENG solves Eq. (3) via a repeated tangent space approximation to the manifold for each time step.

TENG subroutine within each time step. The key idea of TENG is shown in Fig. 1. During each time step, TENG minimizes the loss function in *multiple iterations* (similar to OBTI), and within each iteration, it updates the parameters based on the function u-space gradient (of the loss function) *projected to the parameter θ-space* (similar to TDVP). Here, we reserve the phrase “time step” for physical time steps of the PDE and “iteration” for optimization steps within each physical time step.

The details of TENG iterations within a single time step are shown in Subroutine `TENG_stepper`, where α_n is the learning rate at the nth iteration, and the `least_square`$(J_{(x),j}, \Delta u(x))$ should be interpreted as solving the least square problem

$$\Delta\theta = \underset{\Delta\theta\in\mathbb{R}^{N_p}}{\arg\min} \left\| \Delta u(\cdot) - \textstyle\sum_j J_{(\cdot),j}\Delta\theta_j \right\|^2_{L^2(\mathcal{X})}. \quad (6)$$

Subroutine `TENG_stepper`

```
Input: $\theta_{\text{init}}, u_{\text{target}}$
$n \leftarrow 0, \theta \leftarrow \theta_{\text{init}}$
while $n < N_{\text{it}}$ do
    $\Delta u(x) \leftarrow -\alpha_n \frac{\partial L(\hat{u}_\theta, u_{\text{target}})}{\partial \hat{u}_\theta}(x)$
    $J_{(x),j} \leftarrow \frac{\partial \hat{u}_\theta(x)}{\partial \theta_j}$
    $\Delta\theta \leftarrow \texttt{least\_square}(J_{(x),j}, \Delta u(x))$
    $\theta \leftarrow \theta + \Delta\theta$
    $n \leftarrow n + 1$
end while
Output: $\theta$
```

We note that when Subroutine `TENG_stepper` is performed under certain approximations, it can be reduced to TDVP or OBTI (see Appendix A for detail).

TENG resolves the limitations of both TDVP and OBTI. As mentioned in Sec. 3.2, TDVP suffers from inaccurate tangent space approximation for nonzero Δt. TENG does not suffer from this issue because of the repeated use of tangent space projections, which eventually minimizes Eq. (3) on the manifold. This also gives the following theorem as a direct consequence, which does not hold for TDVP.

Theorem 4.1. *The optimal solution of TENG is reparameterization invariant even with nonzero Δt.*

Proof. TENG achieves its optimum when $\hat{u} \in \mathcal{M}_\Theta$ is closest to u_{target} at each time step. Since a reparameterization does not change the manifold $\mathcal{M}_\Theta$ and the loss is defined in the function space, therefore, the optimal solution differs by merely a relabeling of parameters at the same point in $\mathcal{M}_\Theta$. □

The global convergence of natural gradient has been studied in certain non-linear neural network (Zhang et al., 2019). Although achieving global optimal is not theoretically guaranteed in general, in practice, the loss function is usually convex in the u-space, and $\hat{u}_\theta$ is often close to u_{target} because of small time step sizes. The result is likely to be close to global optimal. In practice, we observe the optimization can result in loss values close to machine precision ($\mathcal{O}(10^{-14})$).

In addition, while TDVP may require solving an ill-conditioned least square equation and an inaccurate solution directly affects the θ-space ODE, solving the least square problem is only part of the optimization procedure for TENG, which turns out to have a smaller side effect. An inaccurate least square solution does not lead to an inaccurate solution to Eq. (3), given sufficient iterations. The resulting loss value of Eq. (3) also provides a concrete metric for TENG on the accuracy during optimization.

As discussed in Sec. 3.3, the main challenge for the current OBTI method lies in the difficulty of optimizing the time integrating loss function (Eq. (3)). While the loss function is a complicated non-convex function in the parameter θ-space, it is usually convex in the u-space; therefore, it is advantageous to perform gradient descent in u-space and project the solution to θ-space. Furthermore, TENG can also benefit from the reparametrization invariant property described above. While an efficient higher-order time integration method is still lacking in the current OBTI method, in this work we show how to incorporate higher-order methods into TENG.

TENG formulation over time steps. The most simple time integration scheme is the forward Euler's method, which only keeps the lowest order Taylor expansion of the PDE. When integrated in the TENG method, we set $u_{\text{target}} = \hat{u}_{\theta_t} + \Delta t \mathcal{L}\hat{u}_{\theta_t}$ and use Subroutine `TENG_stepper` to solve for $\hat{u}_{\theta_{t+\Delta t}}$. The full algorithm is summarized in Algorithm `TENG_Euler`.

Algorithm `TENG_Euler`: A 1st-order integration scheme

```
**Input:** $\theta_{t=0}$, $\Delta t$, $T$
$t \leftarrow 0$
**while** $t < T$ **do**
    $u_{\text{target}}(x) \leftarrow \hat{u}_{\theta_t}(x) + \Delta t \mathcal{L}\hat{u}_{\theta_t}(x)$
    $\theta_{t+\Delta t} \leftarrow$ TENG_stepper$(\theta_t, u_{\text{target}})$
    $t \leftarrow t + \Delta t$
**end while**
**Output:** $\theta_{t=T}$
```

Algorithm `TENG_Heun`: A 2nd-order integration scheme

```
**Input:** $\theta_{t=0}$, $\Delta t$, $T$
$t \leftarrow 0$
**while** $t < T$ **do**
    $u_{\text{temp}}(x) \leftarrow \hat{u}_{\theta_t}(x) + \Delta t \mathcal{L}\hat{u}_{\theta_t}(x)$
    $\theta_{\text{temp}} \leftarrow$ TENG_stepper$(\theta_t, u_{\text{temp}})$
    $u_{\text{target}}(x) \leftarrow \hat{u}_{\theta_t}(x) + \frac{\Delta t}{2}\left(\mathcal{L}\hat{u}_{\theta_t}(x) + \mathcal{L}u_{\theta_{\text{temp}}}(x)\right)$
    $\theta_{t+\Delta t} \leftarrow$ TENG_stepper$(\theta_{\text{temp}}, u_{\text{target}})$
    $t \leftarrow t + \Delta t$
**end while**
**Output:** $\theta_{t=T}$
```

Beyond the first-order Euler's method, Algorithm `TENG_Heun` provides an example of applying second-order integration method. In this method, an intermediate target solution u_{temp} is used, and a set of intermediate parameters θ_{temp} is trained. The intermediate parameters are used to construct u_{target} and $\theta_{t+\Delta t}$. Our method avoids terms like $\mathcal{L}^n \hat{u}_{\theta_t}$ or $\partial \mathcal{L}\hat{u}_{\theta_t}/\partial\theta_t$ that often appear in existing OBTI methods (Donatella et al., 2023; Luo et al., 2022; 2023), reducing the cost and improving numerical stability.

Connection to natural gradient. We note that the algorithm outlined in Subroutine `TENG_stepper` can be reformulated using the conventional Hilbert natural gradient in the form

$$\Delta\theta_i = -\alpha \sum_j G^{-1}(\theta)_{i,j} \frac{\partial L(\hat{u}_\theta, u_{\text{target}})}{\partial \theta_j}(x), \qquad (7)$$

with $G(\theta)$ the Hilbert gram matrix (see Appendix A for detail). However, solving least square equations is more stable, with the added flexibility of choosing least square solvers. Therefore, we use the formulation in Subroutine `TENG_stepper` for practical implementation.

Alternatively, Subroutine `TENG_stepper` can also be viewed as a generalized Gauss–Newton method. Therefore, TENG can also be interpreted as the abbreviation of time-evolving *Newton–Gauss* (also see Appendix A).

4.2. Complexity and Error Analysis

The computational complexity of TENG from $t = 0$ to T is $\mathcal{O}(C_{\text{lstsq}} N_{\text{it}} T/\Delta t)$, where $C_{\text{lstsq}} = \mathcal{O}(N_s N_p^2)$ (with N_s the number of samples and N_p the number of parameters) is the cost of solving the least square equation in each iteration, N_{it} is the number of iterations in each time step, and $T/\Delta t$ is the number of physical time steps. In comparison, the computational complexity of TDVP is $\mathcal{O}(C'_{\text{lstsq}} T/\Delta t)$ and the computational complexity of OBTI is $\mathcal{O}(N'_{\text{it}} T/\Delta t)$. Although the cost of TENG includes both C_{lstsq} and N_{it}, both of the terms can be *significantly smaller* than those in TDVP and OBTI due to the following reasons.

In TDVP, the quality of the least square solution directly corresponds to the accuracy; therefore the least square equation must be solved with high accuracy and thus require high cost. Even with sparse update (Berman & Peherstorfer, 2023), one may not be able to use too few parameters; otherwise, the update may be inaccurate. In contrast, the least square equation can be solved approximately in TENG without compromising accuracy. In this work, we design a sparse update scheme for TENG. In each iteration, we randomly sub-sample parameters and only solve the least square equation within the space of these parameters, which significantly helps reduce the cost.

OBTI, on the other hand, requires a large number of iterations in every time step to minimize the loss function, due to the difficulty of the non-convex optimization landscape. In contrast, in our TENG method, the loss values decrease to close to machine precision with only $\mathcal{O}(1)$ iterations (see Sec. 5.2 and Appendix B for detail). In practice, we observe that TENG is able to improve the accuracy by orders of magnitude while keeping a similar computational cost to TDVP and OBTI (also see Appendix B).

The error of TENG is in general determined by (i) the expressivity of the neural network (ii) the optimization algorithm (iii) the time integration scheme. Based on the universal approximation theorem, with a proper choice of neural network, it is likely that the neural network is sufficiently powerful to represent the underlying solution at every time step; thus the error from (i) is small in general. Our TENG algorithm is able to achieve loss values close to machine precision at every time step; therefore the error from (ii) error is also small. Given sufficiently small errors in (i) and (ii), factor (iii) dominates the convergence property of TENG. At the same time, higher-order time-integration schemes can be integrated with TENG, in which case the error from (iii) follows the standard numerical analysis results for solving differential equations. From the above perspectives, we further contrast TENG with other algorithms. While TDVP does not have an optimization error, the projection step already introduces some errors, which can be severe for nonzero time step sizes and when the least square equation in Eq. (2) is low rank (Berman & Peherstorfer, 2023). For OBTI, the error from (ii) can be large, resulting in poor performances, in addition to the lack of efficient higher-order time-integration schemes in prior works.

TENG also permits error estimation based on the decomposition of errors above. Below, we outline the error estimation for TENG-Euler. Errors for TENG with higher-order integration methods can be estimated analogously.

Let $\varepsilon_p(t)$ be the L^p-error between the TENG-Euler solution and the exact solution at time t, $\varepsilon_p^{\mathrm{EE}}(t)$ between the exact solution and the solution evolved exactly according to the Euler's method, $\varepsilon_p^{\mathrm{TE}}(t)$ between the TENG-Euler and the solution evolved exactly according to the Euler's method, $r(\cdot,t)$ the residual function after the TENG-Euler optimization of the time step at t, and $\mathcal{G} := 1 + \Delta t\mathcal{L}$.

Theorem 4.2. *The error $\varepsilon_p(t)$ is bounded by $\varepsilon_p^{\mathrm{EE}}(t) + \varepsilon_p^{\mathrm{TE}}(t)$, where $\varepsilon_p^{\mathrm{EE}}(t)$ is an order $\mathcal{O}(\Delta t)$ quantity, and $\varepsilon_p^{\mathrm{TE}}(t) = \left\|\sum_{n=0}^{t/\Delta t-1}\mathcal{G}^n r(\cdot, t-n\Delta t)\right\|_{L^p(\mathcal{X})}$.*

Proof. See Appendix Theorem A.2. □

5. Experiments

5.1. Equations and Setup

Heat equation. The first example we choose is the two-dimensional isotropic heat equation

$$\frac{\partial u}{\partial t} = \nu\left(\frac{\partial^2 u}{\partial x_1^2} + \frac{\partial^2 u}{\partial x_2^2}\right) \tag{8}$$

with a diffusivity constant $\nu = 1/10$. The heat equation describes the physical process of heat flow or particle diffusion in space. Since it permits an analytical solution in the frequency domain, the heat equation is an ideal test bed for benchmarking (see Appendix D for details).

Allen–Cahn equation. We also consider Allen–Cahn equation

$$\frac{\partial u}{\partial t} = \nu\left(\frac{\partial^2 u}{\partial x_1^2} + \frac{\partial^2 u}{\partial x_2^2}\right) + u - u^3 \tag{9}$$

with a diffusivity constant $\nu = 1/200$, which is a reaction-diffusion model that describes the process of phase separation. The Allen–Cahn equation is nonlinear and does not permit analytical solutions in general. In addition, its solution usually involves sharp boundaries and can be challenging to solve numerically. As a benchmark, we solve it using a spectral method (Canuto et al., 2007) (see Appendix D for detail) and consider its solution to be a *proxy* of the exact solution as the reference.

Burgers' equation. We further benchmark our method on the viscous Burgers' equation

$$\frac{\partial u}{\partial t} = \nu\left(\frac{\partial^2 u}{\partial x_1^2} + \frac{\partial^2 u}{\partial x_2^2}\right) - u\left(\frac{\partial u}{\partial x_1} + \frac{\partial u}{\partial x_2}\right) \tag{10}$$

with a diffusivity (viscosity) constant $\nu = 1/100$. In Appendix B, we also explore cases with smaller ν. Burgers' equation is a convection-diffusion equation that describes phenomena in various areas, such as fluid mechanics, nonlinear acoustics, gas dynamics, and traffic flow. This equation can generate sharp gradients that propagate over time, especially for small ν, which can be challenging to solve. Similar to the Allen–Cahn equation, Burgers' equation does not have a general analytical solution either. Therefore, we also use the spectral method (Canuto et al., 2007) solution as a *proxy* of the exact solution as the reference (see Appendix D for detail).

PDE domain, boundary, and initial condition. For all three equations, we first benchmark on two spatial dimensions in the domain $\mathcal{X} = [0, 2\pi) \times [0, 2\pi)$ and $\mathcal{T} = [0, 4]$, with periodic boundary condition and the following initial condition

$$\begin{aligned} u_0(x_1, x_2) = \frac{1}{100}\Big(&\exp\big(3\sin(x_1) + \sin(x_2)\big) \\ &+ \exp\big(-3\sin(x_1) + \sin(x_2)\big) \\ &- \exp\big(3\sin(x_1) - \sin(x_2)\big) \\ &- \exp\big(-3\sin(x_1) - \sin(x_2)\big)\Big) \end{aligned} \tag{11}$$

This initial condition is anisotropic, contains peaks and valleys at four different locations, and consists of many frequencies besides the lowest frequency, which can result in challenging dynamics for various PDEs.

For the Heat equation, we in addition consider a challenging three-dimensional benchmark, where we again choose periodic boundary conditions in the domain $\mathcal{X} = [0, 2\pi)^3$ and $\mathcal{T} = [0, 8]$. The initial condition is chosen to be a combination of sinusoidal terms in the following form so the exact solution can be analytically calculated.

$$\begin{aligned} u_0(x_1, x_2, x_3) = A_{000} &+ \sum_{k_1=1}^{2}\sum_{k_2=1}^{2}\sum_{k_3=1}^{2} A_{k_1k_2k_3}\prod_{i=1}^{3}\cos(k_i x_i) \\ &+ \sum_{k_1=1}^{2}\sum_{k_2=1}^{2}\sum_{k_3=1}^{2} B_{k_1k_2k_3}\prod_{i=1}^{3}\sin(k_i x_i), \end{aligned} \tag{12}$$

where the coefficients are randomly chosen (see Appendix C for coefficients used in this work). In Appendix C, we also explore the heat equation on a 2D disk $\mathcal{X} = \mathcal{D}_2 = \{(x_1, x_1) : x_1^2 + x_2^2 \leq 1\}$.

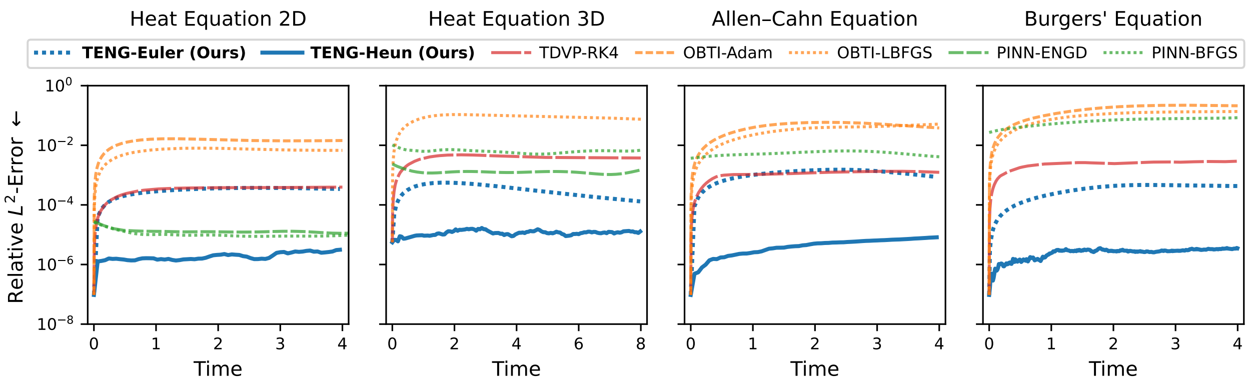

Figure 2. Benchmark of TENG, in terms of relative L^2-error as a function of time, against various algorithms on two- and three-dimensional heat equations, Allen–Cahn equation and Burgers' equation. All sequential-in-time methods use the same time step size $\Delta t = 0.005$ for heat and Allen–Cahn equations and $\Delta t = 0.001$ for Burgers' equation.

For the Burgers' equation, cases with unequal domains and additional initial conditions are also explored in Appendix C.

Baselines. While TENG is sequential-in-time, our benchmarks include both sequential-in-time (TDVP and OBTI) and global-in-time (PINN) methods. For TDVP, we choose the recently proposed sparse update method (Berman & Peherstorfer, 2023), which has been shown to outperform previous full update methods. In addition, we use the same fourth-order Runge–Kutta integration scheme. For the OBTI method, we choose the standard Euler's integration scheme with $L(\hat{u}_\theta, u_{\text{target}}) = \|\hat{u}_\theta - u_{\text{target}}\|^2_{L^2(\mathcal{X})}$ (the same loss as TENG). Both Adam and L-BFGS optimizers are used as benchmarks. For all sequential-in-time methods, we use the same time step $\Delta t = 5 \times 10^{-3}$ for the heat equation. For Allen–Cahn equation and Burgers' equation, we first compare all sequential-in-time methods with $\Delta t = 5 \times 10^{-3}$ and $\Delta t = 1 \times 10^{-3}$ respectively, before analyzing the effect of various Δt. In addition, All sequential-in-time methods share the same neural network architecture and initial parameters at $t = 0$. For PINN, we test both BFGS and the recently proposed ENGD optimizer. Since Ref. (Müller & Zeinhofer, 2023) did not provide the implementation for Allen–Cahn equation and Burgers' equation, we omit the benchmark of ENGD optimizer for the two equations. We use a network architecture similar to Ref. (Müller & Zeinhofer, 2023) (see Appendix E for detail).

Error metric. We consider the following two error metrics:

1. relative L^2-error at each time step

$$\varepsilon(t) = \frac{\|\hat{u}(\cdot, t) - u_{\text{reference}}(\cdot, t)\|_{L^2(\mathcal{X})}}{\|u_{\text{reference}}(\cdot, t)\|_{L^2(\mathcal{X})}}, \tag{13}$$

2. global relative L^2-error integrated over all time steps

$$\varepsilon_g = \frac{\|\hat{u}(\cdot, \cdot) - u_{\text{reference}}(\cdot, \cdot)\|_{L^2(\mathcal{X}\times\mathcal{T})}}{\|u_{\text{reference}}(\cdot, \cdot)\|_{L^2(\mathcal{X}\times\mathcal{T})}}, \tag{14}$$

where $u_{\text{reference}}$ refers to the analytical solution for the heat equation, and the spectral method solution for Allen–Cahn and Burgers' equation (see Appendix D for detail).

5.2. Results

Benchmark against other methods. We start the benchmark of our method against other methods described in Sec. 5.1 in terms of both the relative L^2-error (Eq. (13)) as a function of time (Fig. 2) and the global relative L^2-error (Eq. (14)) integrated over all time (Table 1).

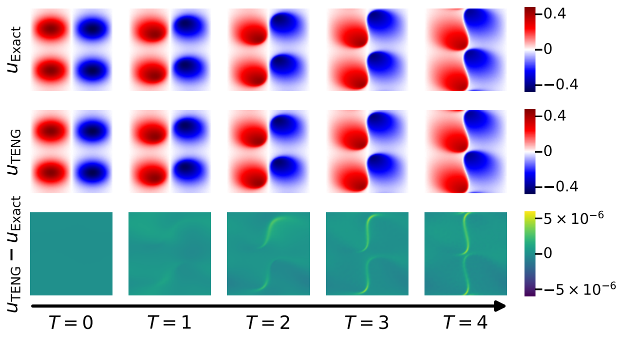

Figure 3. Reference solution, TENG solution, and the difference between them for Burgers' equation. The reference solution is generated using the spectral method, and the TENG solution shown here uses the TENG-Heun method with $\Delta t = 0.001$.

In all cases, our TENG-Heun method achieves results orders of magnitude better compared to other methods. Upon

Method	Global Relative L^2-Error ↓			
	Heat (2D)	Heat (3D)	Allen–Cahn	Burgers'
TENG-Euler (Ours)	3.006×10^{-4}	*3.664×10^{-4}*	*1.249×10^{-3}*	*3.598×10^{-4}*
TENG-Heun (Ours)	$\mathbf{1.588 \times 10^{-6}}$	$\mathbf{1.139 \times 10^{-5}}$	$\mathbf{6.187 \times 10^{-6}}$	$\mathbf{2.643 \times 10^{-6}}$
TDVP-RK4	3.279×10^{-4}	3.841×10^{-3}	1.258×10^{-3}	2.437×10^{-3}
OBTI-Adam	1.391×10^{-2}	–	4.966×10^{-2}	1.696×10^{-1}
OBTI-LBFGS	6.586×10^{-3}	8.743×10^{-2}	4.180×10^{-2}	1.047×10^{-1}
PINN-ENGD	1.403×10^{-5}	2.846×10^{-3}	–	–
PINN-BFGS	*1.150×10^{-5}*	1.389×10^{-2}	5.540×10^{-3}	6.538×10^{-2}

Table 1. Benchmark of TENG, in terms of global relative L^2-error, against various algorithms on the heat equation, Allen–Cahn equation and Burgers' equation. The best result in each column is marked in boldface and the second best result is marked in italic font. Here, the same Δt as in Fig. 2 is used.

closer inspection, our TENG method with Euler's integration scheme is already comparable to or better than the TDVP method with the RK4 integration scheme. In addition, TENG-Euler is significantly better than OBTI with both Adam and L-BFGS optimizers, both of which use the same integration scheme. In Fig. 3, We show the difference between TENG-Heun and the reference solution by plotting the function evolution over time. It can be seen that our method traces closely with the reference solution with a tiny deviation on the order of $\mathcal{O}(10^{-6})$. In Appendix B, we show additional details of runtime, and the relation between runtime and performance.

Convergence speed and machine precision accuracy. We further demonstrate the convergence of TENG-Euler in Fig. 4 compared to OBTI-Adam and OBTI-LBFGS. In a single time step, TENG achieves a training loss value (with the squared L^2 distance as the loss function) close to machine precision $\mathcal{O}(10^{-14})$ with only a few iterations, while OBTI can only get to $\mathcal{O}(10^{-7})$ loss after a few hundred iterations. We also plot the final loss of each time step optimization and show that TENG stably reaches the machine precision for all time, which is seven orders of magnitude better than OBTI. Our results have shown the high accuracy of TENG compared to the existing approaches (see Appendix B for additional results). Since the final loss values are near machine precision for all time steps, we believe the main error source of TENG-Euler comes from the Euler's expansion, instead of the neural network approximation. This is further verified later during time integration scheme comparisons.

Compare time integration schemes. We further examine the effects of time integration schemes on TENG and compare our TENG-Euler and TENG-Heun methods with different time step sizes.

In Fig. 5, we show the global relative L^2-error (defined in Eq. (14)) as a function of time step size Δt. It can be seen from the figure that while TENG-Euler already achieves a global relative L^2-error of $\mathcal{O}(10^{-4})$ for small Δt, using a

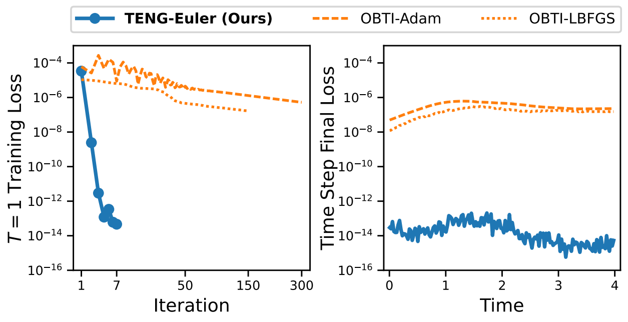

Figure 4. Training loss during the time step at $T = 1$ and final training losses for all time steps for the TENG-Euler method and the two OBTI methods for Allen–Cahn equation.

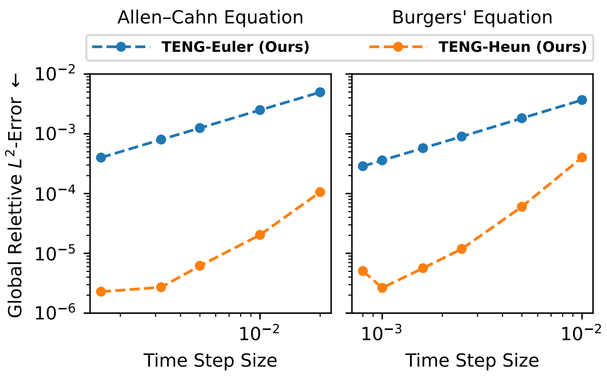

Figure 5. Comparison of different time integration schemes of TENG with respect to the time step sizes on Allen–Cahn equation and Burgers' equation, using global relative L^2-error as a metric.

higher order integration scheme significantly reduces the error to $\mathcal{O}(10^{-6})$. In addition, TENG-Heun can maintain the low error even at relatively large Δt, signifying the advantage of our implementation of higher-order integration schemes. We note that, for small time step sizes, the accumulation of per-step error dominates, while for large time step sizes, the discretization error from the integration scheme dominates, resulting in the TENG-Heun with the smallest Δt not as good as larger Δt. In addition, the curves for TENG-Euler and TENG-Heun have different slopes. Both phenomena are consistent with numerical analysis results for traditional PDE solvers. Additional explorations with TENG-RK4 method can be found in Appendix B.

6. Discussion and Conclusion

We introduce *Time-Evolving Natural Gradient*, a novel approach that generalizes time-dependent variational principles and optimization-based time integration, resulting in a highly accurate and efficient PDE solver utilizing natural gradient. TENG, encompassing algorithms like TENG-Euler and advanced variants such as TENG-Heun, significantly outperforms existing state-of-the-art methods in accuracy, achieving machine precision in solving a range of PDEs. For future work, it would be interesting to explore the application of TENG in more diverse and complex real-world scenarios, particularly in areas where traditional PDE solutions are currently unfeasible. While this work is focused on two- and three-dimensional (spatial) scalar-valued PDEs with periodic boundary conditions, the same method can be considered for generalizing to vector-valued PDE in other numbers of dimensions, and other boundary conditions, such as the Dirichlet boundary condition or the Neumann boundary condition. It will also be important to develop TENG for broader classes of PDEs besides initial value problems with applications to nonlinear and multi-scale physics PDEs in various domains. Advancing TENG's integration with cutting-edge machine learning architectures and optimizing its performance for large-scale computational tasks will be a vital area of research for computational science and engineering.

Acknowledgements

The authors acknowledge support from the National Science Foundation under Cooperative Agreement PHY-2019786 (The NSF AI Institute for Artificial Intelligence and Fundamental Interactions, http://iaifi.org/). This material is based upon work supported by the U.S. Department of Energy, Office of Science, National Quantum Information Science Research Centers, Co-design Center for Quantum Advantage (C2QA) under contract number DE-SC0012704. This material is also in part based upon work supported by the Air Force Office of Scientific Research under the award number FA9550-21-1-0317. The research was sponsored by the United States Air Force Research Laboratory and the Department of the Air Force Artificial Intelligence Accelerator and was accomplished under Cooperative Agreement Number FA8750-19-2-1000. The authors acknowledge the MIT SuperCloud and Lincoln Laboratory Supercomputing Center for providing (HPC, database, consultation) resources that have contributed to the research results reported within this paper. Some computations in this paper were run on the FASRC cluster supported by the FAS Division of Science Research Computing Group at Harvard University.

Impact Statement

Through the advancement of the *Time-Evolving Natural Gradient* (TENG), which solves partial differential equations (PDEs) with enhanced accuracy and efficiency, our work exhibits broader impact spans multiple disciplines, including but not limited to, climate modeling, fluid dynamics, materials science, and biomedical engineering. While the primary goal of this work is to push forward computational techniques within the field of machine learning, it inherently carries the potential for significant societal benefits, such as improved environmental forecasting models, more efficient engineering designs, and advancements in medical technology. Ethically, the deployment of TENG should be approached with consideration to ensure that enhanced computational capabilities translate into positive outcomes without unintended consequences given its accuracy and reliability in real-world applications. There are no specific ethical concerns that we feel must be highlighted at this stage; however, we acknowledge the importance of ongoing evaluation of the societal and ethical implications as this technology is applied. This acknowledgment aligns with our commitment to responsible research and innovation, understanding that the true value of advancements in machine learning is realized through their contribution to societal progress and well-being.

References

Amari, S.-I. Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276, 1998.

Berman, J. and Peherstorfer, B. Randomized sparse neural galerkin schemes for solving evolution equations with deep networks. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

Canuto, C., Hussaini, M. Y., Quarteroni, A., and Zang, T. A. *Spectral methods: evolution to complex geometries and applications to fluid dynamics*. Springer Science & Business Media, 2007.

Carleo, G. and Troyer, M. Solving the quantum many-body problem with artificial neural networks. *Science*, 355 (6325):602–606, 2017.

Chen, H., Wu, R., Grinspun, E., Zheng, C., and Chen, P. Y. Implicit neural spatial representations for time-dependent pdes. In *International Conference on Machine Learning*, pp. 5162–5177. PMLR, 2023a.

Chen, R. T., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

Chen, Z., Luo, D., Hu, K., and Clark, B. K. Simulating 2+1d lattice quantum electrodynamics at finite density with neural flow wavefunctions. *arXiv preprint arXiv:2212.06835*, 2022.

Chen, Z., Newhouse, L., Chen, E., Luo, D., and Soljacic, M. Antn: Bridging autoregressive neural networks and tensor networks for quantum many-body simulation. In Oh, A., Neumann, T., Globerson, A., Saenko, K., Hardt, M., and Levine, S. (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 450–476. Curran Associates, Inc., 2023b.

Dirac, P. A. Note on exchange phenomena in the thomas atom. In *Mathematical proceedings of the Cambridge philosophical society*, volume 26, pp. 376–385. Cambridge University Press, 1930.

Donatella, K., Denis, Z., Le Boité, A., and Ciuti, C. Dynamics with autoregressive neural quantum states: Application to critical quench dynamics. *Physical Review A*, 108(2), August 2023. ISSN 2469-9934. doi: 10.1103/physreva.108.022210.

Du, Y. and Zaki, T. A. Evolutional deep neural network. *Physical Review E*, 104(4), October 2021. ISSN 2470-0053. doi: 10.1103/physreve.104.045303.

Gutiérrez, I. L. and Mendl, C. B. Real time evolution with neural-network quantum states. *Quantum*, 6:627, 2022.

Han, J., Jentzen, A., et al. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Communications in mathematics and statistics*, 5 (4):349–380, 2017.

Han, J., Jentzen, A., and E, W. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34): 8505–8510, 2018.

Kakade, S. M. A natural policy gradient. *Advances in neural information processing systems*, 14, 2001.

Koch, O. and Lubich, C. Dynamical low-rank approximation. *SIAM Journal on Matrix Analysis and Applications*, 29(2):434–454, 2007.

Kochkov, D. and Clark, B. K. Variational optimization in the ai era: Computational graph states and supervised wavefunction optimization. *arXiv preprint arXiv:1811.12423*, 2018.

Langley, P. Crafting papers on machine learning. In Langley, P. (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., and Anandkumar, A. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.

Long, Z., Lu, Y., Ma, X., and Dong, B. Pde-net: Learning pdes from data. In *International conference on machine learning*, pp. 3208–3216. PMLR, 2018.

Lu, L., Jin, P., and Karniadakis, G. E. Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *arXiv preprint arXiv:1910.03193*, 2019.

Luo, D., Chen, Z., Carrasquilla, J., and Clark, B. K. Autoregressive neural network for simulating open quantum systems via a probabilistic formulation. *Physical review letters*, 128(9):090501, 2022.

Luo, D., Chen, Z., Hu, K., Zhao, Z., Hur, V. M., and Clark, B. K. Gauge-invariant and anyonic-symmetric autoregressive neural network for quantum lattice models. *Physical Review Research*, 5(1):013216, 2023.

Müller, J. and Zeinhofer, M. Achieving high accuracy with pinns via energy natural gradient descent. In *International Conference on Machine Learning*, pp. 25471–25485. PMLR, 2023.

Pascanu, R. and Bengio, Y. Revisiting natural gradient for deep networks. *arXiv preprint arXiv:1301.3584*, 2013.

Peters, J., Vijayakumar, S., and Schaal, S. Reinforcement learning for humanoid robotics. In *Proceedings of the third IEEE-RAS international conference on humanoid robots*, pp. 1–20, 2003.

Pfaff, T., Fortunato, M., Sanchez-Gonzalez, A., and Battaglia, P. W. Learning mesh-based simulation with graph networks. *arXiv preprint arXiv:2010.03409*, 2020.

Raissi, M., Perdikaris, P., and Karniadakis, G. E. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019.

Sanchez-Gonzalez, A., Godwin, J., Pfaff, T., Ying, R., Leskovec, J., and Battaglia, P. Learning to simulate complex physics with graph networks. In *International conference on machine learning*, pp. 8459–8468. PMLR, 2020.

Sinibaldi, A., Giuliani, C., Carleo, G., and Vicentini, F. Unbiasing time-dependent Variational Monte Carlo by projected quantum evolution. *Quantum*, 7:1131, October 2023. ISSN 2521-327X. doi: 10.22331/q-2023-10-10-1131.

Sirignano, J. and Spiliopoulos, K. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375:1339–1364, 2018.

Stokes, J., Izaac, J., Killoran, N., and Carleo, G. Quantum natural gradient. *Quantum*, 4:269, 2020.

Wang, J., Chen, Z., Luo, D., Zhao, Z., Hur, V. M., and Clark, B. K. Spacetime neural network for high dimensional quantum dynamics. *arXiv preprint arXiv:2108.02200*, 2021a.

Wang, S. and Perdikaris, P. Long-time integration of parametric evolution equations with physics-informed deeponets. *Journal of Computational Physics*, 475:111855, 2023.

Wang, S., Wang, H., and Perdikaris, P. Learning the solution operator of parametric partial differential equations with physics-informed deeponets. *Science advances*, 7(40): eabi8605, 2021b.

Weinan, E., Han, J., and Jentzen, A. Algorithms for solving high dimensional pdes: from nonlinear monte carlo to machine learning. *Nonlinearity*, 35(1):278, 2021.

Yu, B. et al. The deep ritz method: a deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, 6(1):1–12, 2018.

Zhang, G., Martens, J., and Grosse, R. B. Fast convergence of natural gradient descent for over-parameterized neural networks. *Advances in Neural Information Processing Systems*, 32, 2019.

A. Additional Theoretical Results

Theorem A.1. *TDVP is not reparameterization invariant with* Δt*.*

Proof. We will construct an explicit counter-example. For simplicity, consider a zero-dimensional PDE (an ODE) $\partial_t u = u$, whose solution is $u = u_0 \exp(t)$. Let $\hat{u}_\theta = \theta$ and $\hat{v}_\phi = \exp\phi$, which are just reparameterizations of each other. In the parameter space, the two ODEs read $\partial_t \theta = \theta$ and $\partial_t \phi = 1$. Let both of them evolve for a discrete time step Δt from $t = 0$, we have $\theta_{\Delta t} = \theta_0 + \Delta t \theta_0$ and $\phi_{\Delta t} = \phi_0 + \Delta t$. Plugging back into the functions, $\hat{u}_{\theta_{\Delta t}} = \hat{u}_{\theta_0} + \Delta t \hat{u}_{\theta_0}$ and $\hat{v} = \hat{v}_{\theta_0} \exp(\Delta t)$ It is evidential that the two parameterizations give different solutions. □

TENG can be reduced to TDVP under certain assumptions. For simplicity, we will be focusing on the first-order Euler's method. Consider Subroutine `TENG_stepper`. Let the loss function $L(\hat{u}_{\theta_t}, u_{\text{target}}) = \|\hat{u}_{\theta_t} - u_{\text{target}}\|^2_{L^2(\mathcal{X})}$ and $N_{\text{it}} = 1$. For simplicity, let $u_{\text{target}} = \hat{u}_{\theta_t} + \Delta t \mathcal{L} \hat{u}_{\theta_t}$ be the first-order Euler expansion. Then,

$$\frac{\partial L}{\partial \hat{u}_{\theta_t}}(x) = 2(\hat{u}_{\theta_t}(x) - u_{\text{target}}(x)) \equiv 2\Delta t \mathcal{L} \hat{u}_{\theta_t}. \tag{A.1}$$

Choosing $\alpha = 1/2$, we have $\Delta u = \Delta t \mathcal{L} \hat{u}_{\theta_t}$. Then, the least square equation becomes

$$\Delta\theta = \underset{\Delta\theta \in \mathbb{R}^{N_p}}{\arg\min} \left\| \Delta t \mathcal{L} \hat{u}_{\theta_t}(\cdot) - \textstyle\sum_j J_{(\cdot),j} \Delta\theta_j \right\|^2_{L^2(\mathcal{X})}, \tag{A.2}$$

which is the same as the TDVP algorithm with nonzero time step sizes.

TENG can be reduced to OBTI under certain assumptions. Let $N_{\text{it}} > 0$. As mentioned in the main paper, approximate methods can be used to solve the least square equation in Subroutine `TENG_stepper`. Here, let its solution be approximated by a single gradient descent, which gives rise to

$$\Delta\theta_j = \int J_{(x),j} \Delta u(x) \mathrm{d}x \equiv -\alpha \int \frac{\partial \hat{u}_\theta(x)}{\partial \theta_j} \frac{\partial L}{\partial \hat{u}_\theta}(x) \mathrm{d}x = -\alpha \frac{\partial L}{\partial \theta_j}, \tag{A.3}$$

which reduces to the regular gradient descent in the θ-space with many iterations.

Hilbert natural gradient formulation of Subroutine `TENG_stepper`. Consider the least square equation

$$\Delta\theta = \underset{\Delta\theta \in \mathbb{R}^{N_p}}{\arg\min} \left\| \Delta u(\cdot) - \textstyle\sum_j J_{(\cdot),j} \Delta\theta_j \right\|^2_{L^2(\mathcal{X})}. \tag{A.4}$$

It's solution is given by the normal equation $J^T J \Delta\theta = J^T \Delta u$ where we use the matrix notation and omit the indices. The solution to the normal equation is given by $\Delta\theta = (J^T J)^{-1} J^T \Delta u$. Notice that

$$(J^T \Delta u)_j = \int J_{(x),j} \Delta u(x) \mathrm{d}x \equiv -\alpha \int \frac{\partial \hat{u}_\theta(x)}{\partial \theta_j} \frac{\partial L}{\partial \hat{u}_\theta}(x) \mathrm{d}x = -\alpha \frac{\partial L}{\partial \theta_j}. \tag{A.5}$$

In addition,

$$(J^T J)_{i,j} = \int \frac{\partial \hat{u}_\theta(x)}{\partial \theta_i} \frac{\partial \hat{u}_\theta(x)}{\partial \theta_j} \mathrm{d}x \equiv G_{i,j}(\theta), \tag{A.6}$$

where $G(\theta)$ is the Hilbert gram matrix (Müller & Zeinhofer, 2023). Therefore, Eq. (A.4) can be written as the Hilbert natural gradient descent

$$\Delta\theta_i = -\alpha \sum_j G^{-1}(\theta)_{i,j} \frac{\partial L(\hat{u}_\theta, u_{\text{target}})}{\partial \theta_j}(x). \tag{A.7}$$

We note that while these two formulations are mathematically equivalent, the least square formulation has a few practical advantages. First, it allows for more stable numerical solvers. In general, the Hilbert gram matrix has a condition number twice as large as the original Jacobian matrix. If the original least square equation is ill-conditioned, Eq. (A.7) is even worse. In addition, when the least square equation is underdetermined, solving the original least square problem gives the minimum norm solution, whereas Eq. (A.7) has to be solved with pseudo-inverse, which can be numerically unstable in practice.

Generalized Gauss–Newton formulation of Subroutine `TENG_stepper`. Let the loss function be the squared L^2-distance. Define $r(x) := \hat{u}_\theta(x) - u_{\text{target}}(x)$.The derivative of loss in function space is given by

$$\frac{\partial L}{\partial \hat{u}_\theta}(x) = 2(\hat{u}_\theta(x) - u_{\text{target}}(x)) \equiv 2r(x). \tag{A.8}$$

In matrix notation, the iteration above becomes

$$\Delta\theta = -\alpha(J^T J)^{-1} J^T r. \tag{A.9}$$

When $\alpha = 1/2$, this reduces to one iteration of the Gauss–Newton method. Therefore, Subroutine `TENG_stepper` can also be viewed as a generalized Gauss–Newton method.

Theorem A.2. *The error $\varepsilon_p(t)$ is bounded by $\varepsilon_p^{\text{EE}}(t) + \varepsilon_p^{\text{TE}}(t)$, where $\varepsilon_p^{\text{EE}}(t)$ is an order $\mathcal{O}(\Delta t)$ quantity, and $\varepsilon_p^{\text{TE}}(t) = \left\| \sum_{n=0}^{t/\Delta t - 1} \mathcal{G}^n r(\cdot, t - n\Delta t) \right\|_{L^p(\mathcal{X})}$.*

Proof. Denote $D(\cdot,t) = u(\cdot,t) - \hat{u}_\theta(\cdot,t) = (u(\cdot,t) - u^{\text{Eu}}(\cdot,t)) + (u^{\text{Eu}}(\cdot,t) - \hat{u}_\theta(\cdot,t)) \equiv D^{EE}(\cdot,t) + D^{TE}(\cdot,t)$, where $u(\cdot,t)$ is the exact solution, $u^{\text{Eu}}(\cdot,t)$ is the solution from Euler method, $\hat{u}_\theta(\cdot,t)$ is the TENG-Euler solution at time t. By definition, $\varepsilon_p(t) = \|D(\cdot,t)\|_{L^p(\mathcal{X})}$, $\left\|u(\cdot,t) - u^{\text{Eu}}(\cdot,t)\right\|_{L^p(\mathcal{X})} = \left\|D^{EE}(\cdot,t)\right\|_{L^p(\mathcal{X})} = \varepsilon^{EE}(t)$, and $\left\|u^{\text{Eu}}(\cdot,t) - \hat{u}(\cdot,t)\right\|_{L^p(\mathcal{X})} = \left\|D^{TE}(\cdot,t)\right\|_{L^p(\mathcal{X})} = \varepsilon^{TE}(t)$. It follows by the triangular inequality that $\varepsilon_p(t) \leq \varepsilon_p^{\text{EE}}(t) + \varepsilon_p^{\text{TE}}(t)$. Since the Euler method is a first-order method, $\varepsilon_p^{\text{EE}}(t)$ has an error of order $O(\Delta t)$.

Denote the optimization error in time t as $r(\cdot,t)$, such that $\hat{u}_\theta(\cdot,t+\Delta t) - \mathcal{G}\hat{u}_\theta(\cdot,t) = r(\cdot,t)$. It follows that $u^{\text{Eu}}(\cdot,t+\Delta t) - D^{TE}(\cdot,t+\Delta t) - \mathcal{G}(u^{\text{Eu}}(\cdot,t) - D^{TE}(\cdot,t)) = r(\cdot,t)$, which implies that $D^{TE}(\cdot,t+\Delta t) = \mathcal{G}D^{TE}(\cdot,t) - r(\cdot,t)$ due to the cancellation of $u^{Eu}(\cdot,t) - \mathcal{G}u^{Eu}(\cdot,t)$ from the exact Euler method. By induction, $\varepsilon_p^{\text{TE}}(t) = \left\| \sum_{n=0}^{t/\Delta t - 1} \mathcal{G}^n r(\cdot, t - n\Delta t) \right\|_{L^p(\mathcal{X})}$.

□

B. Additional Experimental Results

In this section, we show additional benchmark results. In Fig. B.1, we show the training losses of TENG and OBTI methods during the time steps at $T = 0.8, 1.6, 2.4, 3.2, 4.0$ for Allen–Cahn equation. For Euler's integration scheme, we compare our TENG-Euler method with both OBTI-Adam and OBTI-LBFGS algorithms and find that our algorithm consistently achieves loss values orders of orders of magnitudes better than OBTI, with only 7 iterations. For TENG-Heun method, each time step requires training a set of intermediate parameters. Therefore, each figure includes two curves. As shown in the figure, all stages converge to machine precision within a small number of iterations.

In Fig. B.2 B.3 and B.4, we show the density plots for the two-dimensional heat equation, Burgers' equation and Allen–Cahn equation. In each figure, we plot the reference solution (see Appendix D for details on obtaining the reference solution), the TENG-Heun solution, the TDVP-RK4 solution, the OBTI-LBFGS solution and the PINN-BFGS solution, and their difference to the reference solution. In all cases, the TENG-Heun solution closely tracks the reference solution, with a maximum error of order $\mathcal{O}(10^{-6})$, whereas solutions generated by other methods can have relatively larger solutions.

In Fig. B.5, we plot the global relative L^2-error of PINN during training. We show both PINN-ENGD and PINN-BFGS for the heat equation, and PINN-BFGS for Allen–Cahn equation and Burgers' equation. We observe that while PINN-ENGD converges very quickly on the heat equation, PINN-BFGS eventually surpases PINN-ENGD. In addition, Allen–Cahn equation and Burgers' equation appear to be significantly more challenging for PINN, where it finds difficulty optimizing the error to below $\mathcal{O}(10^{-2})$.

In Fig. B.6, we further explore the advantage of higher-order integration schemes. In particular, we plot the relative L^2-error as a function of time for TENG-Euler, TENG-Heun, and TENG-RK4. For the heat equation, TENG-RK4 fails to significantly surpass TENG-Heun, which could be attributed to the error accumulation under small Δt in this case. For the other two equations, we explore larger Δt and find that TENG-RK4 is able to achieve small errors, while TENG-Heun's performance starts to deteriorate.

In Table. B.1, we report the runtime for the runs in Fig. 2. Our TENG methods significantly improve the simulation accuracy with a similar runtime to other algorithms. We note that TENG-Heun is roughly twice as costly as TENG-Euler due to the

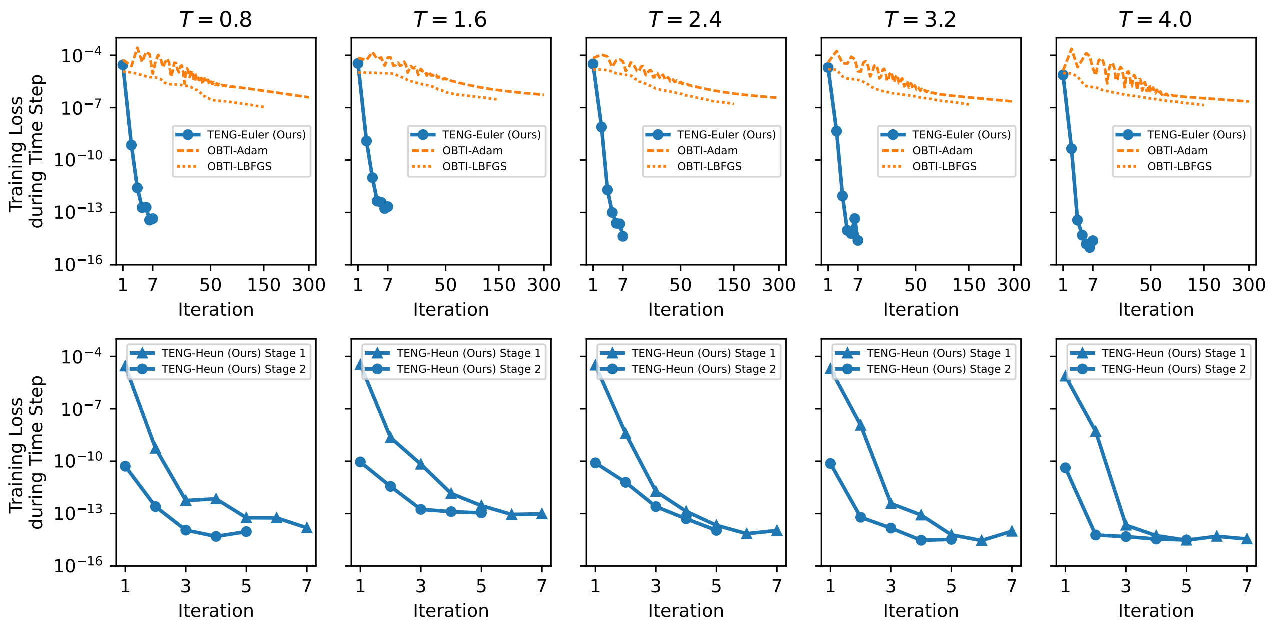

Figure B.1. Training loss during many time steps for TENG-Euler, TENG-Heun, and the two OBTI methods for Allen–Cahn equation with a time step size $\Delta t = 0.005$. TENG-Heun method requires two training stages, one for θ_{temp}, and the other for $\theta_{t+\Delta t}$. Therefore, each figure contains two curves.

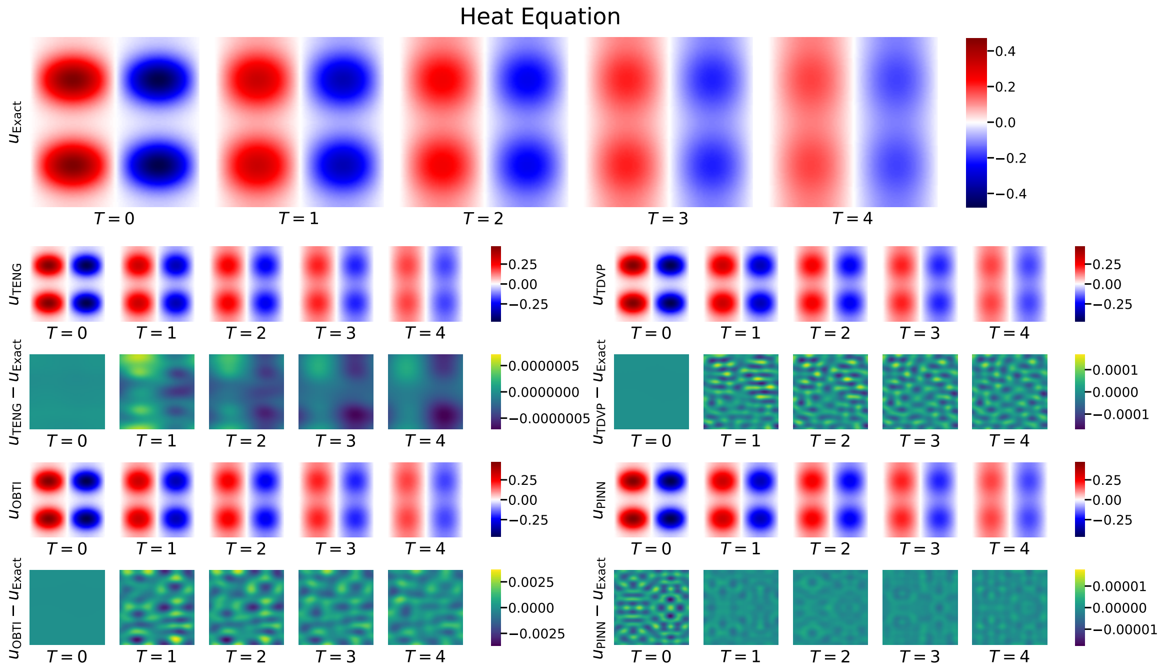

Figure B.2. Exact, TENG, TDVP, OBTI, and PINN solutions and their differences from the reference solution for the two-dimensional heat equation. The reference solution is generated using the analytical solution, the TENG solution shown here uses the TENG-Heun method, the OBTI shown here uses the OBTI-LBFGS method, and the PINN shown here uses the PINN-BFGS method. The error of our TENG method is orders of magnitude smaller than other methods.

two-stage training process in each time step. In addition, all sequential-in-time methods use significantly more time on

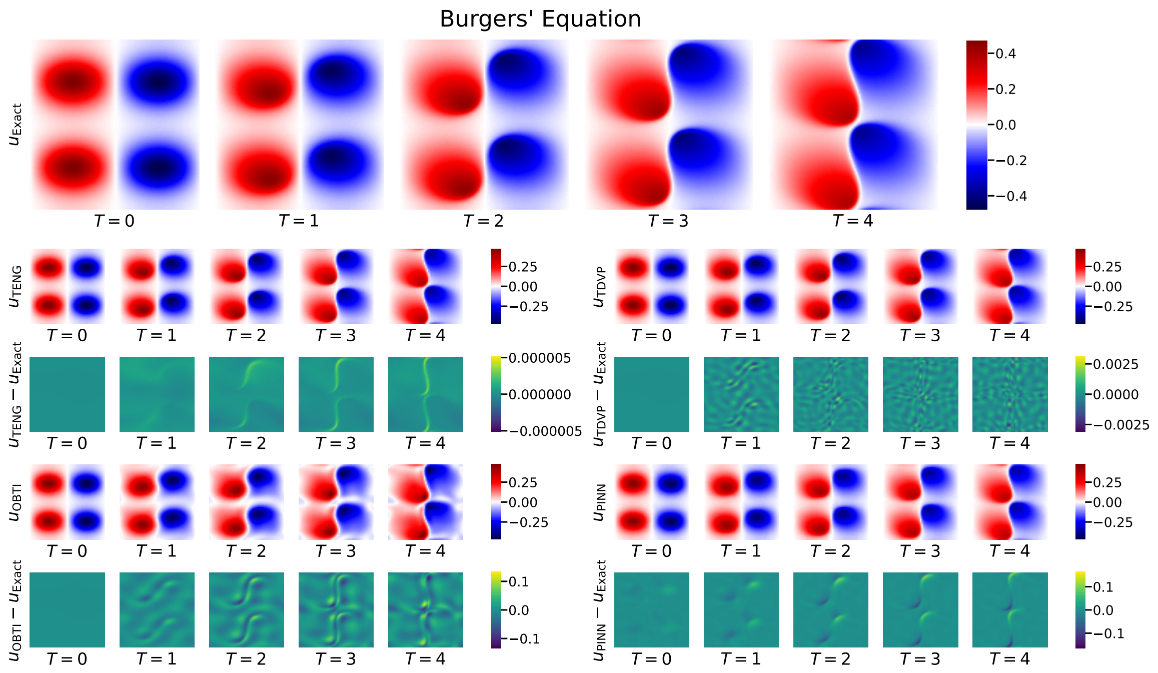

Figure B.3. Exact, TENG, TDVP, OBTI and PINN solutions and their differences from the reference solution for Burgers' equation. The reference solution is generated using the spectral method, the TENG solution shown here uses the TENG-Heun method, the OBTI shown here uses the OBTI-LBFGS method, and the PINN shown here uses the PINN-BFGS method. The error of our TENG method is orders of magnitude smaller than other methods.

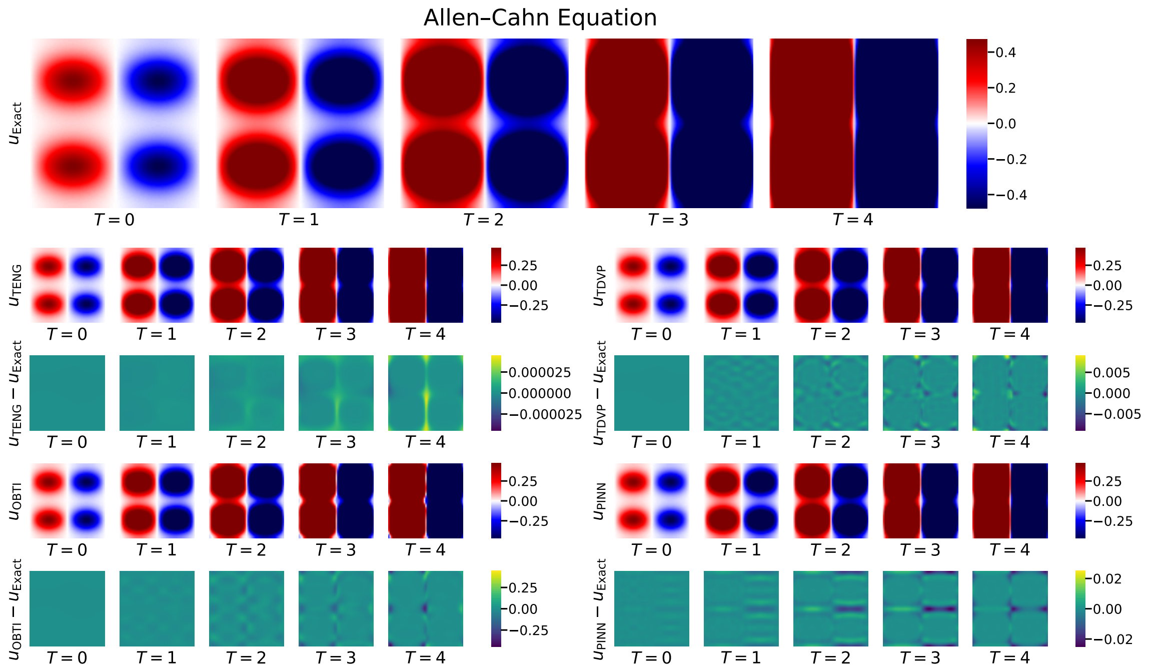

Figure B.4. Exact, TENG, TDVP, OBTI and PINN solutions and their differences from the reference solution for Allen–Cahn equation. The reference solution is generated using the spectral method, the TENG solution shown here uses the TENG-Heun method, the OBTI shown here uses the OBTI-LBFGS method, and the PINN shown here uses the PINN-BFGS method. The error of our TENG method is orders of magnitude smaller than other methods.

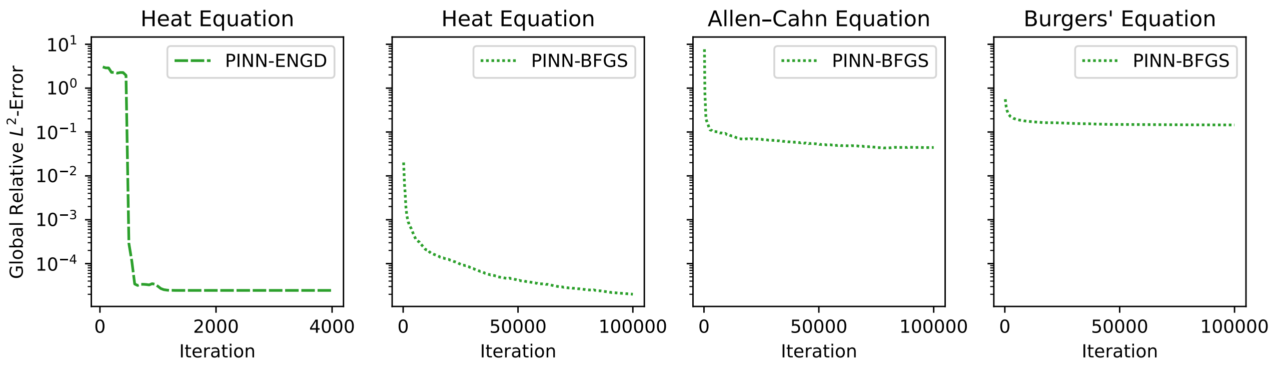

Figure B.5. Global relative L^2-error for PINN as a function of training iterations for the two-dimensional heat equation, Allen–Cahn equation, and Burgers' equation.

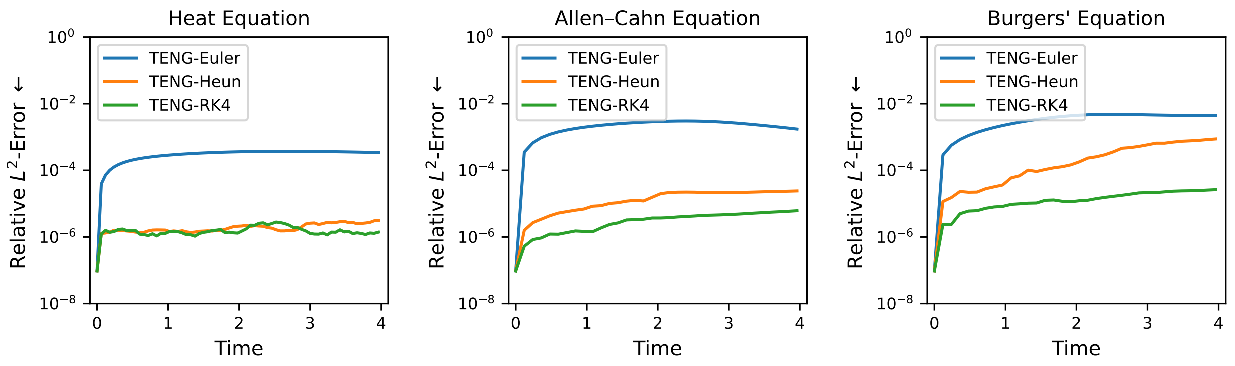

Figure B.6. Relative L^2-error as a function of time for the two-dimensional heat equation ($\Delta t = 0.005$), Allen–Cahn equation ($\Delta t = 0.01$), and Burgers' equation ($\Delta t = 0.01$) for various integration methods.

Burgers' equation, due to the reduced time step Δt. While the result of PINN could benefit from a longer training process for the Burgers' equation, we believe it is unlikely as shown in the training dynamics in Fig. B.5. In Fig. B.7, we plot the global relative L^2-error as a function of runtime, with various choices of hyperparameters listed in Appendix E. The figure shows that TENG achieves significantly lower error compared to other methods, even for low runtimes. (The five points with the highest errors for TENG all use the Euler integration scheme, where the dominant error is the Euler discretization error.) We note that all experiments are performed on a single NVIDIA V100 GPU with 32GB memory. In all cases, the 32GB memory is sufficient for our benchmarks.

Method	Runtime (Hours)		
	Heat	Allen–Cahn	Burgers'
TENG-Euler (Ours)	2.5	2.5	12.7
TENG-Heun (Ours)	4.1	4.2	20.9
TDVP-RK4	4.6	4.4	21.1
OBTI-Adam	3.0	3.2	19.6
OBTI-LBFGS	4.4	4.1	22.1
PINN-ENGD	1.1	–	–
PINN-BFGS	2.0	2.9	3.6

Table B.1. Runtime for various algorithms for the two-dimensional heat equation, Allen–Cahn equation, and Burgers' equation.

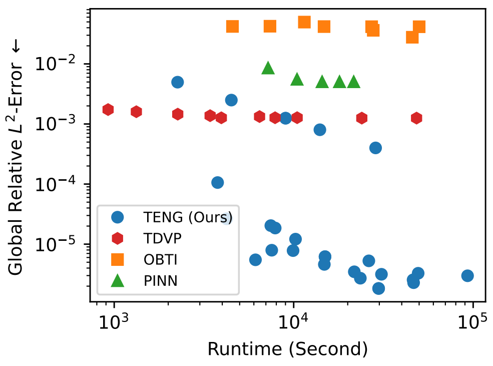

Figure B.7. Global relative L^2-error as a function of runtime for various algorithms under various hyperparameters.

C. Additional Initial Conditions and Benchmarks

As mentioned in the main paper, for the three-dimensional heat equation, we consider a initial condition in the form

$$u_0(x_1,x_2,x_3) = A_{000} + \sum_{k_1=1}^{2}\sum_{k_2=1}^{2}\sum_{k_3=1}^{2}\left(A_{k_1k_2k_3}\prod_{i=1}^{3}\cos\left(k_ix_i\right) + B_{k_1k_2k_3}\prod_{i=1}^{3}\sin\left(k_ix_i\right)\right). \tag{C.10}$$

Here, we choose the following coefficients: $A_{000} = 0.043$ with the rest of $A_{k_1k_2k_3}$'s and $A_{k_1k_2k_3}$'s shown in Table C.2.

$A_{k_1k_2k_3}$	$k_1 = 1$		$k_1 = 2$	
	$k_2 = 1$	$k_2 = 2$	$k_2 = 1$	$k_2 = 2$
$k_3 = 1$	0.047	-0.021	0.034	-0.02
$k_3 = 2$	-0.021	-0.041	0.024	0

$B_{k_1k_2k_3}$	$k_1 = 1$		$k_1 = 2$	
	$k_2 = 1$	$k_2 = 2$	$k_2 = 1$	$k_2 = 2$
$k_3 = 1$	-0.075	-0.056	-0.027	-0.008
$k_3 = 2$	0.074	-0.007	0.032	0

Table C.2. $A_{k_1k_2k_3}$'s and $B_{k_1k_2k_3}$'s for the initial condition of three-dimensional heat equation.

In addition, we consider an example of the heat equation defined on a two-dimensional disk with Dirichlet boundary condition. Here, the boundary condition is enforced via an additional loss term in Eq. (3), and the initial condition is shown below.

$$\begin{aligned} u_0(r,\theta) = \frac{1}{4}\Big(&Z_{01}(r,\theta) - \frac{1}{4}Z_{02}(r,\theta) + \frac{1}{16}Z_{03}(r,\theta) - \frac{1}{64}Z_{04}(r,\theta) \\ &+Z_{11}(r,\theta) - \frac{1}{2}Z_{12}(r,\theta) + \frac{1}{4}Z_{13}(r,\theta) - \frac{1}{8}Z_{14}(r,\theta) + Z_{21}(r,\theta) + Z_{31}(r,\theta) + Z_{41}(r,\theta)\Big), \end{aligned} \tag{C.11}$$

where r and θ is the polar coordinate variables and Z_{mn} represent the disk harmonics defined as

$$Z_{mn}(r,\theta) = J_m(\lambda_{nm}r)\cos(m\theta) \tag{C.12}$$

with J_m the mth Bessel function and λ_{nm} the nth zero of the mth Bessel function. We note that while the analytical solution is solved in the polar coordinates, all neural network based methods solve the equation and benchmark in the original Cartesian coordinates.

For Burgers' equation, we, in addition, consider benchmarks that include a case with smaller $\nu = 3/1000$ with the original domain, boundary, and initial conditions, and a case with $\nu = 1/100$ but with nonequal domain $\mathcal{X} = [0, 2) \times [0, 2\pi)$ with periodic boundary condition, and $\mathcal{T} = [0, 4]$, and the following initial condition.

$$u_0(x_1, x_2) = \frac{1}{50} \exp\left(\cos\left(\pi x_1 - 2\right) + \sin\left(x_2 - 1\right)\right)^2 . \tag{C.13}$$

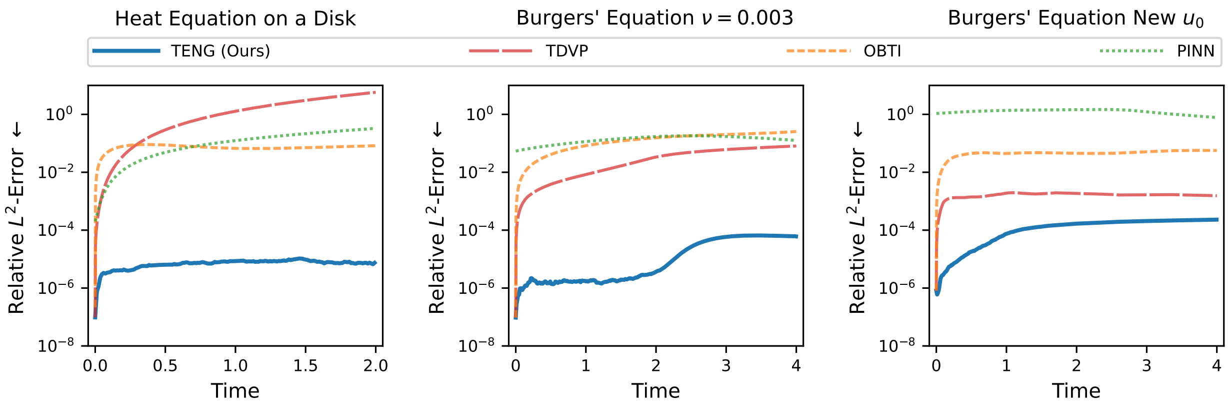

Figure C.8. Relative L^2-error as a function of time for additional benchmarks. For all sequential-in-time methods, we choose time step size $\Delta t = 0.005$ for the heat equation and $\Delta t = 0.001$ for Burgers' equation.

In Fig. C.8, we show the additional benchmarks for the aforementioned examples. Here, TENG refers to the TENG-Heun method, OBTI refers to the OBTI-LBFGS method, and PINN refers to the PINN-BFGS method. We note that for the heat equation on a disk with Dirichlet boundary condition, an additional boundary term is included in the loss function defined in Eq. (3) for TENG and OBTI method. (PINN can also incorporate this boundary term analogously.) However, it is unclear how to enforce the boundary condition in TDVP without redesigning the neural network architecture; therefore, we choose to not enforce the boundary condition for TDVP, which could be the reason why TDVP performs particularly badly on the heat equation on a disk.

D. Details on Obtaining Reference Solutions

Heat equation. As mentioned in the main paper, the heat equation permits an analytical solution in terms of Fourier series. For example, we show the two-dimensional case below.

$$u(x_1, x_2, t) = \sum_{k_1, k_2} \exp\left(-\nu\left(k_1^2 + k_2^2\right) t\right) \tilde{u}_0(k_1, k_2) \exp\left(ik_1 x_1 + ik_2 x_2\right), \tag{D.14}$$

where we omit the terms $2\pi/P$ because in our case $P = 2\pi$. For the two-dimensional case, evaluating the analytical solution is not practical since it is difficult to express our initial condition in Fourier series analytically,

$$\tilde{u}_0(k_1, k_2) = \sum_{x_1, x_2} u_0(x_1, x_2) \exp\left(-ik_1 x_1 - ik_2 x_2\right) \tag{D.15}$$

not to mention calculating an infinite sum of frequencies. Therefore, we choose to evaluate the initial condition on a $2048 \times 2048 = 4194304$ grid. Then, we use the discrete Fourier transform to calculate the initial condition in the Fourier space, before truncating the maximum frequency 48. (The summation contains around $(2 \cdot 48)^2 \approx 9000$ terms in total). For the three-dimensional case and the case where the domain is a disk, since the initial condition is already defined in terms of sinusoidal functions (or Bessel functions), the solution is analytically calculated.

Allen–Cahn equation. Different from the heat equation, Allen–Cahn equation generally does not permit analytical solutions. Therefore, we solve it using the spectral method and consider the solution as a *proxy* for the exact solution as the reference. Here, the basis functions of the spectral method are chosen to be the same Fourier plane waves, so the solution in real space can be written as

$$u(x_1, x_2, t) = \sum_{k_1, k_2} \tilde{u}(k_1, k_2, t) \exp\left(ik_1 x_1 + ik_2 x_2\right). \tag{D.16}$$

When switching from real space to Fourier space, we have

$$\frac{\partial u}{\partial x_j} \rightarrow ik_j\tilde{u} \quad \text{and} \quad uv \rightarrow \tilde{u} \circ \tilde{v}, \tag{D.17}$$

where $\circ$ means convolution. Therefore, the PDE can be rewritten in the Fourier space as

$$\frac{\partial \tilde{u}}{\partial t} = -\nu(k_1^2 + k_2^2)\tilde{u} + \tilde{u} - \tilde{u} \circ \tilde{u} \circ \tilde{u}. \tag{D.18}$$

Here, we choose a maximum frequency cut-off of 128. (Notice that the maximum number of frequencies encountered is $(3 \cdot 2 \cdot 128)^2 \approx 600000$ when calculating the double convolution.) The initial condition is calculated analogous to the case of the heat equation, via a discrete Fourier transform on the $2048 \times 2048 = 4194304$ grid. Then, Eq. (D.18) is solved using the fourth-order Runge–Kutta integration scheme with a time step $\Delta t = 2 \times 10^{-4}$.

Burgers' equation. Analogous to Allen–Cahn equation, Burger's equation does not have a general analytical solution either, except in the case of $\nu = 0$. Therefore, we use the same spectral method used to solve Allen–Cahn equation. Notice that the term $u\partial u/\partial x_j = \partial u^2/\partial x_j$. Therefore, Burgers' equation in the Fourier space reads

$$\frac{\partial \tilde{u}}{\partial t} = -\nu(k_1^2 + k_2^2)\tilde{u} - \frac{i}{2}(k_1 + k_2)\tilde{u} \circ \tilde{u}. \tag{D.19}$$

Here, we choose a maximum frequency cut-off of 192 (with a maximum of around $(2 \cdot 2 \cdot 192)^2 \approx 600000$ terms when calculating the convolution.) The initial condition is calculated in the same way as the heat and Allen–Cahn equation, and Eq. (D.19) is solved using the fourth-order Runge–Kutta integration scheme with a time step $\Delta t = 1 \times 10^{-4}$.

Accuracy of the solutions. In each case, we carefully verify that the number of grid points, the maximum frequency, and the Δt are sufficient to obtain a solution that is accurate to near numerical precision, by varying them over multiple values and observing that the solution converges. We note that the case for Burgers' equation with $\nu = 0.003$ is challenging for the spectral method and the solution may not converge yet, which means the errors we report could be larger than the actual values.

E. Details of Neural Network Architectures and Optimization

All the algorithms used in this work are implemented in JAX and use double precision. Our code is posted on GitHub at `https://github.com/pde-sim/teng`.

Neural network architectures. We choose the same architecture for all sequential-in-time methods, which allows a fair comparison. Our neural network architecture is loosely based on Ref. (Berman & Peherstorfer, 2023) which consists of multiple feedforward layers with tanh activation function as

$$\hat{u}(x) = W_{n_l}\tanh\left(\cdots\tanh\left(W_1 \text{periodic\_embed}(x) + b_1\right)\cdots\right) + b_{n_l}, \tag{E.20}$$

where the periodic embedding function is defined as

$$\text{periodic\_embed}(x) = \text{concatenate}\left(\left[\textstyle\sum_j a_j \cos\left(x_1 + \phi_j\right) + c_j, \sum_j a_j \cos\left(x_2 + \phi_j\right) + c_j\right]\right) \tag{E.21}$$

to explicitly enforce the periodic boundary condition in the neural network. Here, all W, b, a, and c are trainable parameters. Here, we choose $n_l = 7$ layers and $d_h = 40$ hidden dimensions (periodic embedding vector with size 20 for each x_j).

For PINN, we adopt the same architecture from Ref. (Müller & Zeinhofer, 2023) with the addition of periodic embedding. In addition, we increase the hidden dimension to 64 compared to Ref. (Müller & Zeinhofer, 2023) for better expressivity.

In the case of the heat equation on a 2D disk, we simply remove the periodic embedding layer.

Optimization methods. For TENG, we randomly sub-sample trainable parameters when solving the least square problems. This can be viewed as a regularization method when the original least square problem is ill-conditioned and can significantly reduce the computational cost. During each time step, we randomly sub-sample 1536 parameters in the first iteration and sub-sample 1024 parameters in the subsequent iterations. In TENG-Euler, the neural network is optimized for 7 iterations in each time step; in TENG-Heun, the neural network is optimized for 7 iterations to obtain θ_{temp}, followed by 5 iterations for $\theta_{t+\Delta t}$. We reduce the number of iterations in the second stage because θ_{temp} already gives a good initialization for $\theta_{t+\Delta t}$.

For TDVP, we use the sparse update method proposed by Ref. (Berman & Peherstorfer, 2023), which is also a random sub-sample of parameters for each TDVP step, and has been shown to significantly improve the result compared to a full update of a smaller neural network. Here, we randomly sub-sample 2560 parameters at each time step so that the computational cost of TDVP at each time step roughly matches that of TENG (over the training iterations within each time step).

For OBTI, we compare our method with both the Adam optimizer and the L-BFGS optimizer. Within each time step, the neural network is optimized for 300 iterations when using the Adam optimizer, and 150 iterations when using the L-BFGS optimizer. The Adam optimizer uses an initial learning rate of 1×10^{-5} and an exponential scheduler that decays the learning rate by $1/2$ by the end of the 300 iterations.

For all sequential-in-time methods, we need to train the initial parameters to match the initial conditions. Here we use the same initial parameters for a fair comparison. The initial parameters are trained by first minimizing the loss function

$$L(\hat{u}_\theta, u_0) = \|\hat{u}_\theta - u_0\|^2_{L^2(\mathcal{X})} + \left\|\frac{\partial \hat{u}_\theta}{\partial x_1} - \frac{\partial u_0}{\partial x_1}\right\|^2_{L^2(\mathcal{X})} + \left\|\frac{\partial \hat{u}_\theta}{\partial x_2} - \frac{\partial u_0}{\partial x_2}\right\|^2_{L^2(\mathcal{X})} \tag{E.22}$$

using natural gradient descent, where we use the least square formulation as mentioned in the main paper and (approximately) solve the least square problem using CGLS method, until the loss value decays below 1×10^{-7}. Then, we switch the loss function to

$$L(\hat{u}_\theta, u_0) = \|\hat{u}_\theta - u_0\|^2_{L^2(\mathcal{X})} \tag{E.23}$$

and use the random sub-sample version of the natural gradient descent, with 1536 parameters updated for each iteration until the loss value decays to near machine precision (1×10^{-14}). The L^2-norm in both stages are integrated on a 2D grid of 1024 points in each dimension (around 1000000 points in total).

For PINN, both the initial condition and the time evolution are optimized simultaneously; therefore, it does not use the initial parameters mentioned above. In addition, all the time steps of PINN are optimized simultaneously, instead of step by step. For the optimization, we test the BFGS optimizer, and the recently proposed ENGD optimizer (Berman & Peherstorfer, 2023). We note that the ENGD optimizer requires custom implementation for individual PDEs. Since Ref. (Berman & Peherstorfer, 2023) did not provide the implementation for Allen–Cahn equation and Burgers' equation, we omit the benchmark of ENGD optimizer for the two equations. We train the neural network for 100000 iterations when using the BFGS optimizer, and 4000 iterations when using the ENGD optimizer.

For Fig B.7, the results include various hyperperameters. For all sequential-in-time methods, we include different time step sizes $\Delta t = 0.0016$, 0.0032, 0.005, 0.01 and 0.02. For TENG, we include TENG-Euler, TENG-Heun, and TENG-RK4 with different numbers of iterations (within each time step) ranging from 2 to 20 and different numbers of randomly subselected parameters for solving least squares (within each iteration) ranging from 384 to 2048; for TDVP, we include different numbers of randomly subselected parameters for solving least square projections ranging from 384 to 2560 (where we reach the memory limit of V100 GPU); for OBTI, we include both OBTI-Adam and OBTI-LBFGS with different numbers of iterations (within each time step) ranging from 150 to 300; and for PINN, we use the BFGS optimizer, and include results of different number of iterations (globally) and different neural network sizes.