Version of Record: https://www.sciencedirect.com/science/article/pii/S0021999122003886
Manuscript_549764d40da6111c49f5570289594d30

Efficient hybrid explicit-implicit learning for multiscale problems

Yalchin Efendiev[*], Wing Tat Leung[†], Guang Lin[‡], Zecheng Zhang [§]

September 14, 2021

Abstract

Splitting method is a powerful method to handle application problems by splitting physics, scales, domain, and so on. Many splitting algorithms have been designed for efficient temporal discretization. In this paper, our goal is to use temporal splitting concepts in designing machine learning algorithms and, at the same time, help splitting algorithms by incorporating data and speeding them up. We propose a machine learning assisted splitting scheme which improves the efficiency of the scheme meanwhile preserves the accuracy. We consider a recently introduced multiscale splitting algorithms, where the multiscale problem is solved on a coarse grid. To approximate the dynamics, only a few degrees of freedom are solved implicitly, while others explicitly. This splitting concept allows identifying degrees of freedom that need implicit treatment. In this paper, we use this splitting concept in machine learning and propose several strategies. First, the implicit part of the solution can be learned as it is more difficult to solve, while the explicit part can be computed. This provides a speed-up and data incorporation for splitting approaches. Secondly, one can design a hybrid neural network architecture because handling explicit parts requires much fewer communications among neurons and can be done efficiently. Thirdly, one can solve the coarse grid component via PDEs or other approximation methods and construct simpler neural networks for the explicit part of the solutions. We discuss these options and implement one of them by interpreting it as a machine translation task. This interpretation of the splitting scheme successfully enables us using the Transformer since it can perform model reduction for multiple time series and learn the connection between them. We also find that the splitting scheme is a great platform to predict the coarse solution with insufficient information of the target model: the target problem is partially given and we need to solve it through a known problem which approximates the target. Our machine learning model can incorporate and encode the given information from two different problems and then solve the target problems. We conduct four numerical examples and the results show that our method is stable and accurate.

1 Introduction

Splitting methods are used for many applications, where splitting physics, scales, domains, and so on, are considered. These approaches allow simplifying problems and solving them separately and then coupling. Recently, novel splitting approaches are proposed for multiscale problems. The main idea of these approaches is to identify the coarse grid components of the solution and treat them implicitly, while treating other components explicitly. In particular, the stability

[*]Department of Mathematics, Texas A&M University, College Station, TX 77843, USA & North-Eastern Federal University, Yakutsk, Russia
[†]Department of Mathematics, University of California, Irvine, USA
[‡]Department of Mathematics and Mechanical Engineering, Purdue University
[§]Department of Mathematics, Purdue University

© 2022 published by Elsevier. This manuscript is made available under the Elsevier user license
https://www.elsevier.com/open-access/userlicense/1.0/

analysis shows that one needs to choose spaces appropriately such that the first space contains all components associated with high contrast and multiscale features of the solution. Using this splitting, we show that one can choose the time step that scales as the coarse mesh size and the time step is independent of the contrast.

This approach, on the one hand, allows simplifying the temporal discretization and using explicit schemes with some minimal time step constraints. On the other hand, these approaches allow designing neural network architectures and/or using them to speed up the solutions. Before discussing it, we briefly describe some main ideas of splitting algorithms.

We assume that the forward problem is given by

$$u_t = f(u),$$

where $f(u)$ is a multiscale differential operator. To achieve a contrast-independent stability, splitting $u = u_1 + u_2$ is performed, where u_1 is a coarse-grid approximation and uses much fewer degrees of freedom, while u_2 is other degrees of freedom. To have a stability, one needs certain conditions on the space for u_2, such as $\|u\|_{L^2}/\|u\|_a$ is bounded below independent of the contrast, where $\|\cdot\|_a$ is a norm associated with the operator f (roughly speaking). To achieve this, one needs to include all high-contrast multiscale features in u_1. This construction is described in [12] and uses GMsFEM and NLMC concepts developed in [9, 13]. Other multiscale methods have been developed for various applications [17, 19, 20, 5, 6, 7, 10, 16, 8, 9, 13, 15, 18, 25, 26, 1, 3], which we do not mention. In this paper, we will focus on NLMC and GMsFEM.

One of the contributions of this paper is the use of machine learning to accelerate the computations and discuss a possible design for machine learning [4, 31, 2, 23, 22]. We again comment that the computation of u_1 is more computationally difficult compared to u_2 as it uses implicit discretization. For this reason, we propose the following strategies.

- 1) We learn and predict u_1 by designing appropriate machine learning techniques, while we compute u_2 based on PDEs.
- 2) We design a hybrid neural network, one for u_1 and the other for u_2, and take advantage of the fact that u_2 is explicit.
- 3) We compute u_1 based on data, reduced-order models, ,.. and learn u_2 due to its sparse forward map structure.

Next, we briefly discuss these cases. For the first case, our motivation stems from the fact that the computation of u_1 is expensive as it is implicit and nonlinear (in general). Thus, for its computation, machine learning techniques can be used to predict u_1, while we compute u_2 since its computation is explicit. More precisely, we can train a machine learning model based on a small portion of the entire solution in time and then alternatively predict u_1 and compute u_2.

For the second case, our main idea is to design a hybrid neural network. The computation for u_2 uses sparse information and needs fewer parameters for identification; however, if the problem is nonlinear, one needs to construct the multiscale basis space for u_1 and u_2 at each iteration. This increases the computational cost in computing u_2 by the splitting scheme; hence training and predicting u_2 is a great option.

For the third case, our motivation comes from the fact that u_1 can be observed or extracted from the solution dynamics (e.g., approaches like DMD); on the other hand, the evolution of u_2 uses sparse information and it may not be difficult to train a machine learning model to predict u_2. We then can use approximate dynamics (learned or a priori formulated) to compute u_2.

In this paper, we mostly focus on the first approach. In particular, we use some special neural network architectures for it, which are based on the Transformer. Transformer is a machine learning structure which is originally designed to handle language tasks [27]. It can be regarded as a machine learning model which can encode the long time dependency of the time series [14, 27, 28, 30, 4]. One hence can apply it to perform model reduction in time [4]. Our problem is to predict the dynamics of u_1 or u_2 which are of high dimension, so it is natural to use Transformer to learn the time dependency meanwhile performing the model reduction. Furthermore, Transformer can be used in handling the machine translation task, i.e., it can deal with the problems which have inputs of different space (languages). This motivates us using Transformer to learn u_1 because the dynamics of u_1 depends on both u_1 and u_2 which are in different spaces. We conduct four experiments by using Transformer as the key of the machine learning model and the results show that the prediction is stable and is accurate. We note that one can use different machine learning strategies with our proposed methods (e.g., [24, 21, 29]) and we do not discuss it here.

Another contribution of this work is the coarse solution prediction with incomplete information; that is, we find that the splitting scheme is a good platform to find the coarse solution of a model, but only the partial information of the target model is given. For simplicity, we will call this 'coarse solution assimilation' in the remaining of this work. There are many real life problems whose solution v (different from u) can be decomposed as v_1+v_2, where v_1 can capture the main features of the model while v_2 is the correction. In the real scenario, only incomplete information of v_1 is given and information about v_2 is missing. If one wants to recover v, one way is to solve a model which approximates the target model, i.e., we know the solution $\tilde{v} = \tilde{v}_1 + \tilde{v}_2$ of a model which is closed to the target. The problems like this can be referred to the coarse solution problem. This problem is not easy to solve, but we realize that the splitting scheme can be formulated as a coarse solution assimilation process and we can use our proposed machine learning method to solve the problem easily.

As we have discussed before, u_1 is a coarse-grid approximation and can capture all high-contrast multiscale features. This can be regarded as v_1 in the coarse solution assimilation; moreover, u_2 uses other degree of freedom and is the correction v_2. Now the question is how to solve the problem, i.e., find all $u_1 + u_2$ if it is only given an incomplete u_1 and $\tilde{u}_2$ which comes from an approximating model.

We find that our machine learning model is able to handle this problem. We can train a model which predicts the future u_1 with a small portion of given u_1 and $\tilde{u}_2$. The difficulty is to handle inputs which originate from different spaces (u_1 and $\tilde{u}_2$) and also learn the time dependency. This includes encoding the u_1, $\tilde{u}_2$ and combining u_1 and $\tilde{u}_2$. We find that our proposed method can handle this problem and give a reliable prediction. It should be noted that the strategies mentioned previously can all be incorporated into the coarse solution assimilation. For the demonstration purposes, we will only consider the first case.

In the paper, we present several numerical results. We solve both linear and nonlinear examples and their applications in coarse solution assimilation. We observe that the solutions are stable and both L_2 and energy relative errors are small.

The rest of the paper is organized as follows. In Section 2, we briefly review the splitting scheme, the motivation of the machine learning and the basic machine learning model are introduced in Section 3. We present our proposed method in Section 4 and introduce the splitting-based coarse solution assimilation in Section 4.3. Finally, in Section 5, we verify our proposed method by conducting 4 numerical experiments. The results are also shown in this section.

2 Preliminaries

In this section, we will review the partially explicit splitting scheme [12, 11] which is designed to tackle the multiscale high contrast problems. The scheme is based on the multiscale finite element methods [9, 13] and the stability of the scheme is independent of the contrast ratio of the problem. We consider the following equation

$$u_t = -f(u). \tag{1}$$

In general, we can assume that $f = \dfrac{\delta F}{\delta u}$, the variational derivative of energies $F(u) = \int E_1(u)$, and is contrast dependent and nonlinear (or linear) (i.e., f introduces stiffness in the system). We refer to [11] for more details about assumptions on F for well-posedness, weak formulation, and stability.

To solve the problem, a standard method is finite element approach. We can consider the numerical solution $u_H(t,\cdot) \in V_H$ satisfies

$$(u_{H,t}, v) = -(f(u), v) \; \forall v \in V_H, \tag{2}$$

where V_H is a finite element space and $(\cdot,\cdot)$ denotes the inner product in V_H.

We consider the solution splitting and split V_H as a direct sum of two subspace $V_{H,1}$ and $V_{H,2}$, namely, $V_H = V_{H,1} \oplus V_{H,2}$. In practice, $V_{H,1}$ and $V_{H,2}$ are the coarse multiscale finite element spaces. The finite element solution is then satisfying

$$\begin{aligned}
(u_{H,1,t} + u_{H,2,t}, v_1) + (f(u_{H,1} + u_{H,2}), v_1) = 0 \; \forall v_1 \in V_{H,1},\\
(u_{H,1,t} + u_{H,2,t}, v_2) + (f(u_{H,1} + u_{H,2}), v_2) = 0 \; \forall v_2 \in V_{H,2},
\end{aligned}$$

where $u_H = u_{H,1} + u_{H,2}$. For the time discretization, we consider the temporal splitting introduced in [12] for linear problems and in [11] for nonlinear problems. That is, we can use a partially explicit time discretization. For example, we can consider

$$\begin{aligned}
&(\frac{u_{H,1}^{n+1} - u_{H,1}^{n}}{\tau} + \frac{u_{H,2}^{n} - u_{H,2}^{n-1}}{\tau}, v_1) + (f(u_{H,1}^{n+1} + u_{H,2}^{n}), v_1) = 0 \; \forall v_1 \in V_{H,1},\\
&(\frac{u_{H,1}^{n} - u_{H,1}^{n-1}}{\tau} + \frac{u_{H,2}^{n+1} - u_{H,2}^{n}}{\tau}, v_2) + (f(\omega u_{H,1}^{n+1} + (1-\omega) u_{H,1}^{n+1} + u_{H,2}^{n}), v_2) = 0 \; \forall v_2 \in V_{H,2},
\end{aligned}$$

where τ is the time step and $\omega \in [0,1]$ is a constant. For the linear case, the splitting scheme is

$$(u_{H,1}^{n+1}, v) = (u_{H,1}^{n}, v) - (u_{H,2}^{n} - u_{H,2}^{n-1}, v) - \tau a(u_{H,1}^{n+1} + u_{H,2}^{n}, v), v \in V_{1,H}, \tag{3}$$

$$(u_{H,2}^{n+1}, v) = (u_{H,2}^{n}, v) - (u_{H,1}^{n} - u_{H,1}^{n-1}, v) - \tau a\big((1-\omega)u_{H,1}^{n} + \omega u_{H,1}^{n+1} + u_{H,2}^{n}, v\big), v \in V_{2,H}, \tag{4}$$

where $a(\cdot,\cdot)$ is the bilinear form associated with f. One example for f is the parabolic problem, i.e., $f(u) = \nabla \cdot (\kappa(x)\nabla u)$, where κ is the contrast permeability field. The bilinear form is then given by $a(u,v) = \int_\Omega \kappa(x)\nabla u \nabla v dx$ for $u, v \in V_H$ and Ω is the domain of the problem. For the stability and convergence concerns of the scheme, we refer to [12, 11].

3 Machine learning

3.1 Motivation

One of the great benefits of the splitting scheme is the contrast independent property; however, it doubles the computations since we need to iterate twice in time. Machine learning is widely used

in solving mathematics problems. Compared to the implementation of splitting numerically, the deep learning approach may be more computational friendly. In particular, researchers need to solve a matrix inverse problem in the first equation of the splitting scheme (4); however, one only needs to evaluate a sequence of matrix multiplications in the deep learning framework.

Motivated by this, we propose the deep learning aided splitting scheme. The basic idea is to first compute $\{u_{H,1}^n, u_{H,2}^n\}_{n=1}^M$, where $M << N$ and N is the total number of time steps in time; then we can train machine learning models to predict either $u_{H,1}^n$ or $u_{H,2}^n$, where $n = M+1, ..., N$. We hence propose three strategies.

1. Train a model to predict $u_{H,1}^n$ while computing the $u_{H,2}^n$ according to the splitting scheme. The motivation of this option is that $u_{H,1}^n$ is implicit and is not easy to compute.

2. Train a model to predict $u_{H,2}^n$ while computing $u_{H,1}^n$ by some other methods. The motivation of this idea is that the evolution of $u_{H,2}^n$ is explicit and should be easier to learn.

3. Train two models to predict both $u_{H,1}^n$ and $u_{H,2}^n$. Learning both solutions is also beneficial in particular when one needs to solve a nonlinear problem.

We have discussed the benefits of the three approaches before in the Introduction and in this work, we will focus and elaborate on the first strategy, i.e., we will introduce the machine learning model and discuss the coarse solution assimilation idea based on this strategy in section (4.3). The other two strategies also rely on machine learning models and our proposed model, training and testing methods can be applied directly.

Now, let us dive into the first model problem. More precisely, we are going to train a model which predicts $u_{H,1}^{n+1}$ using a small portion of the computation data in $[0, t]$ where $t << T$; then in the remaining greater computational domain $[t, T]$, we will apply the trained model to predict $u_{H,1}^{n+1}$; this value is then used in the second equation to obtain $u_{H,2}^{n+1}$. It should be noted that the proposed scheme is built based on the splitting scheme (4); our model mimics the first equation and predicts $u_{H,1}^{n+1}$; for $u_{H,2}^{n+1}$ evaluation, we still follow the second equation since it is also a forward process and does not involve intense computation. The proposed method can be summarized as follows.

1. Compute $u_{H,1}^n$ and $u_{H,2}^n$ up to $n = M$, where $M << N$ and N is the total simulation step. The collection $\{u_{H,1}^n, u_{H,2}^n\}_{n=1}^M$ will be denoted as $\mathcal{D}$.

2. Train a model model $\mathcal{N}$ which predicts $u_{H,1}^{n+1}$ using the training dataset $\mathcal{D}$. More precisely, $\mathcal{N}$ is a map from $\left\{\{u_{H,2}^{n-i+1}\}_{i=1}^{n_e}, \{u_{H,1}^{n+i-1}\}_{i=1}^{n_d}\right\}$ to $u_{H,1}^{n+1}$, where n_e and n_d are the input dimension of the encoder and decoder. Theoretically, $n_e = 2$ and $n_d = 2$, we will detail this in the next section.

3. From M to N, predict $u_{H,1}^{n+1}$ by the model $\mathcal{N}$ and compute $u_{H,2}^{n+1}$ by the equation iteratively until the terminal time.

3.2 Self attention and Transformer

We find that the splitting scheme is similar to the machine language translation task in nature. We will apply Transformer to design the machine learning model N. In this section, let us first review the attention and Transformer.

Self attention. Self attention [27, 4] is a model to handle time series data. The input is a time series consisting of multiple time steps. The module can be regarded as a model reduction

[4] and will output a new series with a reduced dimension. It encodes a long context into a smaller size representation [14]. In particular, this encoding is able to relate different positions of this sequence [27] and concentrate on the input's most relevant parts to perform the task [28]. The response at a time step position is the weighted sum of the features (feature: reduced order model for a single time step) at all positions the long range, multi-level dependencies can be enforced [30]. Self-attention has proven to be useful for tasks such as machine reading and learning sentence representations [28].

Transformer. Transformer is an architecture for transforming one sequence into another one (eg. from English to Germany). It is built on the self-attention module and likes the most transduction models, it consists of an encoder and a decoder which are the stack of self attention modules. Please check Figure (1) and refer to [27] for the details of Transformer.

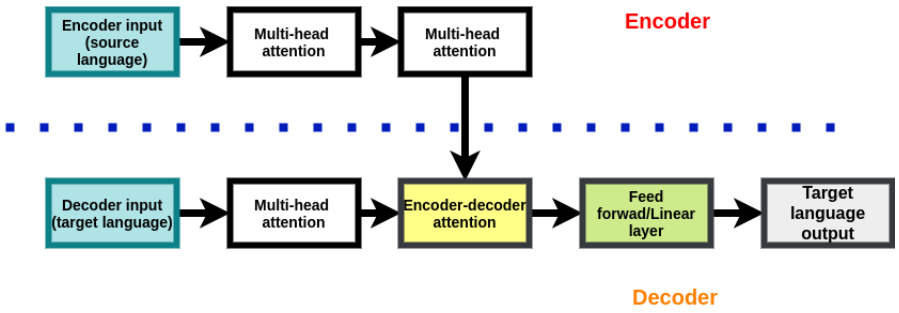


Figure 1: Transformer. The top of the image is the encoder and the bottom is the decoder. Multi-head attention (white) is the stack of self-attention blocks. The encoder-decoder attention (yellow) helps the decoder focus on appropriate places in the source input; it is the key to learn the connections between the encoder and decoder. We skip the positional encoding, softmax and other blocks which are not the focus of this work. Please refer to [27] for details.

Transformer is able to build connections between the encoder and decoder input and output [27]. More precisely, the encoder output is connected to the decoder by an encoder-decoder attention block; this block is able to learn the source and target language connections.

We review the inference phase of Transformer. This is an iterative process which is similar to the time evolution of the numerical scheme: the prediction of the next $u^n_{H,1}$ relies on the previous predictions. The testing has the following steps.

1. Input the full encoder sequence (source language, e.g., English); however, the decoder input is an an empty sequence with a beginning token. The output is a sequence. The first element in the output sequence will be the first word of the target sentence (e.g., Germany). This word will also be used in the prediction of the following words in the target sentence.

2. The first word will be inserted into second place of the decoder input sequence. Together with the entire encoder input (the whole English sentence), Transformer will output a new sequence whose the first element will be the second word of the target sentence.

3. Repeat this process until the end token is predicted and one then gets the whole translated sentence.

The translation process is also iterative, i.e., the next word relies on the previous predictions; however, Transformer is capable to handle the cumulative error or correct the mistakes made in the early stage of the prediction. This motivates us to use Transformer to model the splitting scheme.

4 Proposed method

In Section (3.1), we have introduced the implementation of the proposed method; however, we do not give the details about how to incorporate the splitting with Transformer. In this section, we are going to introduce the motivation of choosing Transformer and give the structure of the network.

4.1 Motivation of using Transformer

It is very natural to design a machine learning network based on the splitting scheme directly. As we have discussed before, the computation involved in the second equation is only the matrices multiplications. We hence focus on the first equation and theoretically, we need to approximate the matrix inversion and multiplication by the network. There should be some model design principles and we summarize the principles based on the first equation as follows.

1. **Principle I**. To get $u_{H,1}^{n+1}$, one needs $u_{H,1}^{n+1}$ and $\{u_{H,2}^{n}, u_{H,2}^{n-1}\}$. They come from different space $V_{H,1}$ and $V_{H,2}$, the model should be able to encode two different time series independently and learn the interaction between the corresponding reduced order models.

2. **Principle II**. The input from $V_{H,2}$ has two time steps $\{u_{H,2}^{n}, u_{H,2}^{n-1}\}$, the model should be capable of capture the dependency of the multiple-step input.

3. **Principle III**. Due to the stability concern, the accumulative error should be controlled in a proper way; that is, if we predict $u_{H,1}^{n+1}$ by the model and compute $u_{H,2}^{n+1}$ by the second equation iteratively, the solutions cannot blow up.

Recall the self-attention and Transformer, they satisfy all design principles. $\{u_{H,1}^{n-i+1}\}_{i=1}^{n_e}$ can be regarded as the target language and input of the decoder; and $\{u_{H,2}^{n+i-1}\}_{i=1}^{n_d}$ is the source language and input of the encoder. We then can summarize how Transformer handles the design difficulties as follows.

1. **Principle I.** According to [27], Transformer is able to build the global dependency between the input and output. This is achieved by the encoder-decoder attention which learns the influence of the source language ($\{u_{H,2}^{n-i+1}\}_{i=1}^{n_e}$) on the target language ($\{u_{H,1}^{n+i-1}\}_{i=1}^{n_d}$).

2. **Principle II.** The self attention mechanism built in Transformer is able to learn the dependency and influence among different positions (time steps).

3. **Principle III.** As reviewed before, Transformer is auto-regressive; i.e., it uses the last predicted word ($u_{H,1}^{n+1}$) in the target sentence as the additional input to infer next word ($u_{H,1}^{n+2}$) in the sentence. Transformer has been successful in various language tasks, we hence believe it can deal with the accumulative errors.

4.2 Proposed structure and implementation

We give the details of the network training and testing method in this section. The splitting scheme is similar to the language translation task, we hence use the similar training and testing strategy as the language task.

Structure. The encoder input is $\{u_{H,2}^{n-i+1}\}_{i=1}^{n_e}$, this can be regarded as the source language with length n_e. The decoder input is the $\{u_{H,1}^{n+i-1}\}_{i=1}^{n_d}$, we take this as the target language. The structure of the network is shown in Figure (2) and Figure (3).

Training. During the training, $\{u_{H,2}^{n-i+1}\}_{i=1}^{n_e}$ and $\{u_{H,1}^{n+i-1}\}_{i=1}^{n_d}$ will be inputs; the output of the network will be $\{u_{H,1}^{n+i}\}_{i=1}^{n_d}$. The training of the network is demonstrated in Figure (2).

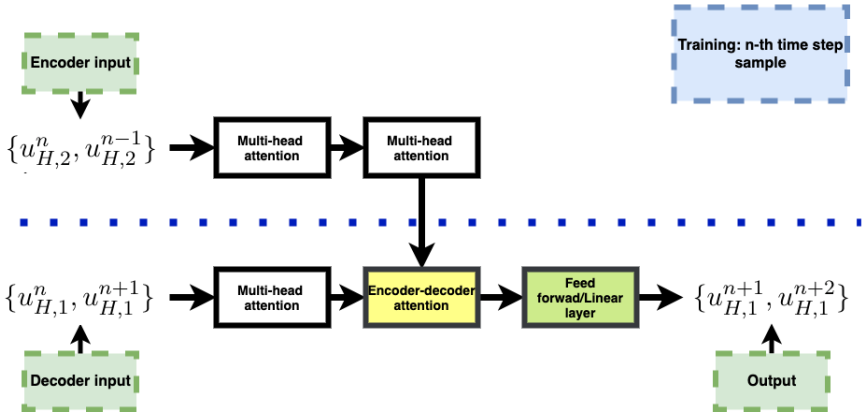


Figure 2: Splitting Transformer model training phase. We show one training sample at time step n. Note that we set $n_d = n_e = 2$, so the model is mimicking the first equation of the splitting scheme.

Testing. This is an iterative process and at the step n, the prediction $u_{H,1}^{n+1}$ of the network will be used in the calculating $u_{H,2}^{n+1}$ in the second equation. We summarize the process as follows and demonstrate the prediction of $u_{H,1}^{n+1}$ in Figure (3).

1. At step n, the encoder input is $\{u_{H,2}^{n-i+1}\}_{i=1}^{n_e}$ and the decoder input is$\{u_{H,1}^{n}, \hat{u}_{H,1}^{n+1}, ..., \hat{u}_{H,1}^{n+n_d-1}\}$; the output of the network is $\{u_{H,1}^{n+1}, \hat{u}_{H,1}^{n+2}, ..., \hat{u}_{H,1}^{n+n_d}\}$.
2. The first entry $u_{H,1}^{n+1}$ will be used in the second equation to calculate $u_{H,2}^{n+1}$; while $\hat{u}_{H,1}^{n+2}, ..., \hat{u}_{H,1}^{n+n_d}$ will be used in predicting $u_{H,1}^{n+2}$ and future $u_{H,1}$ sequence.
3. Set $n = n + 1$ and repeat step 1 and step 2 until reaching the terminal time.

If we follow the splitting scheme, $n_e = 2$ and $n_d = 2$.

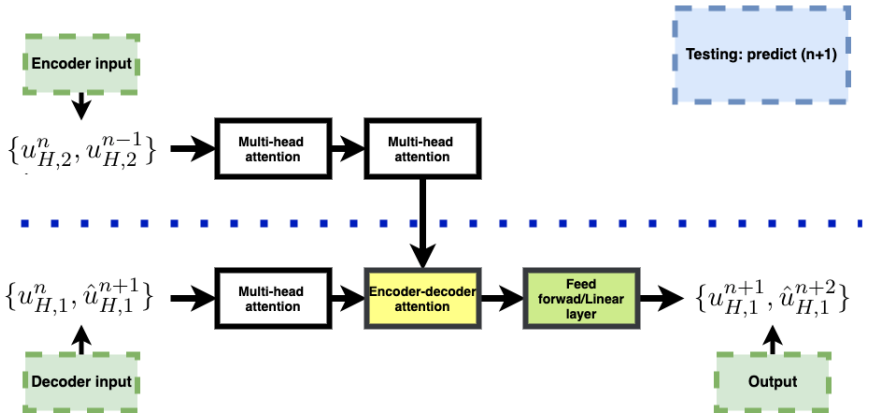


Figure 3: Splitting Transformer model testing phase ($n_e = n_d = 2$). We show the prediction of $u^{n+1}_{H,1}$, which will be used in calculating $u^{n+1}_{H,2}$. The input $\hat{u}^{n+1}_{H,1}$ comes from the second entry of the last step prediction; $\hat{u}^{n+2}_{H,1}$ will be used as the input of the next prediction of $u^{n+2}_{H,1}$. We design the training testing flow in this way to mimic the implicit scheme of the first equation.

Remark. It should be noted that the first equation is implicit in $u^{n+1}_{H,1}$ at step n and our scheme is mimicking the process. At step $n+1$, the target of the prediction is $u^{n+2}_{H,1}$, however, this term has already been predicted in the last iteration n and is used in the input of the decoder at time $n+1$. If $n_d = 2$, denoted as $\hat{u}^{n+2}_{H,1}$, this term is the second entry of the output at step n. We do not use $\hat{u}^{n+2}_{H,1}$ in the calculating of $u^{n+2}_{H,2}$, instead we use $u^{n+2}_{H,1}$ which is the first entry of the output at step $n+1$. Please check Figure (3) for the illustration.

4.3 Coarse solution assimilation

In this section, we will discuss applications of our method. We find that the splitting scheme can be used to solve the target problems; however, a part of the model is implicitly given. For simplicity, we will call this part of the problem the coarse solution assimilation. There are many real life applications whose target model is not given or partially given. Researchers then need to solve the target model based on a known model which approximates the target model. This is an interesting problem and we will explain this model and our method in this section.

We assume that the target solution u_κ of the target equation with permeability $\kappa(x)$ can be decomposed as $u^n_\kappa = u^n_1 + u^n_2, n = 1, ..., N$; where u^n_1 can be regarded as the coarse model while u^n_2 is the correction; n here indicates the time step. We are lack of $\kappa(x)$ information but still want to evaluate u^n_κ, where $n = 1, ..., N$. We claim that if we know $\tilde{\kappa}(x)$ which approximates $\kappa(x)$, and a small piece of $\{u^n_1\}^M_{n=1}$, where $M << N$, then u^n_κ can be recovered by our method. Intuitively, we propose to learn a model that predicts the coarse solution u^n_1 of the target equation; moreover, the coarse solution is automatically corrected by $\tilde{u}^n_2$ from a known equation.

Furthermore, in the framework of the splitting scheme, we have the following discretization for the linear case

$$(u^{n+1}_{H,1}, v) = (u^n_{H,1}, v) - (\tilde{u}^n_{H,2} - \tilde{u}^{n-1}_{H,2}, v) - \tau a(u^{n+1}_{H,1} + \tilde{u}^n_{H,2}, v), v \in V_{1,H}, \tag{5}$$

$$(\tilde{u}^{n+1}_{H,2}, v) = (\tilde{u}^n_{H,2}, v) - (u^n_{H,1} - u^{n-1}_{H,1}, v) - \tau a\big((1-\omega)u^n_{H,1} + \omega u^{n+1}_{H,1} + \tilde{u}^n_{H,2}, v\big), v \in V_{2,H}, \tag{6}$$

where $\tilde{u}_{H,2}$ is the given observation which is the contrast independent component of an approximating solution; this solution may be generated from a permeability $\tilde{\kappa}$ which is close to the target permeability field κ (not given). In this case, we can still predict the coarse grid component $u^n_{H,1}$; and the given $\tilde{u}^n_{H,2}$ can be regarded as the correction of the coarse component solution.

Since $\tilde{u}^n_{H,2}$ is not the real correction of $u^n_{H,1}$, the scheme is not accurate; however, the machine learning can capture the relation between $\tilde{u}^n_{H,2}$ and $u^n_{H,1}$. More precisely, the benefit and the target of machine learning is that the model is supposed to learn and predict all $u^n_{H,1}$ with a small piece of $u^n_{H,1}$ and $\tilde{u}^n_{H,2}$, such that $\tilde{u}^n_{H,2}$ is able to improve $u^n_{H,1}$ and the combined solution $u^n_H = u^n_{H,1}+\tilde{u}^n_{H,2}$ is accurate and stable. In this work, we are going to study this example.

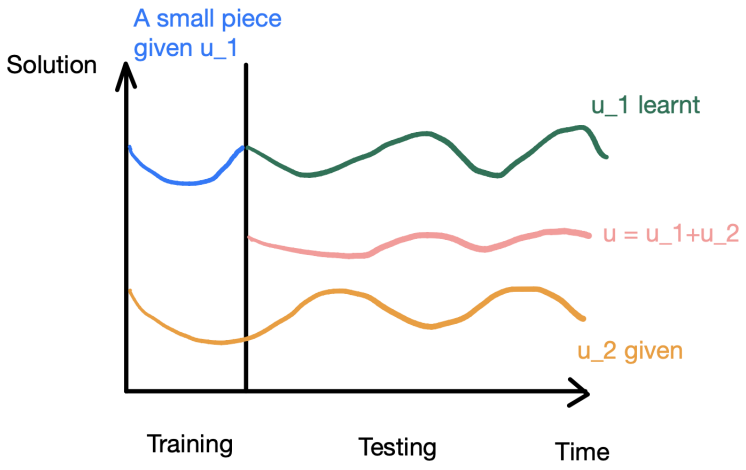


Figure 4: Demonstration of the coarse solution assimilation. We assume a small piece of u_1 and the entire u_2 are given. u_2 may come from a model which approximates the target model. $u = u_1 + u_2$ is the combined solution, in which the learnt u_1 captures the multiscale properties and u_2 is the correction. The machine learning model should be able to learn u_1 basing on u_2 which originates from a different space.

We illustrate the coarse solution assimilation in Figure (4) and to solve this problem, the information we have can be summarized as follows.

1. A permeability $\tilde{\kappa}$ which is closed to $\kappa(x)$. If the approximated permeability is given, the correction solution $\tilde{u}^n_{H,2}$ ($\tilde{\cdot}$ indicates that this is the solution corresponding to $\tilde{\kappa}(x)$) of this permeability can then be evaluated easily.
2. $\{u^n_{H,1}\}_{n=1}^M$ where $M << N$ and $\{\tilde{u}^n_{H,2}\}_{n=1}^N$.

We have one comment about how to obtain $\{\tilde{u}^n_{H,2}\}_{n=1}^N$. The most accurate approach is to compute it by the splitting scheme; however, this sequence can also be trained and predicted. That is, one can train a model $\mathcal{M}$ with $\{\tilde{u}^n_{H,2}\}_{n=1}^M$ where $M << N$ and then predicts the rest of the sequence. This process actually is the experiments designed in section (4) and our numerical experiments show that the prediction is very accurate. Another comment is that our proposed method (training and testing with Transformer) can be used to build this coarse solution assimilation machine learning model. The training and testing (one example is demonstrated in Figure (5)) are similar to the previous setting except that the input of the encoder will be changed to $\{\tilde{u}^n_{H,2}\}_{n=1}^N$.

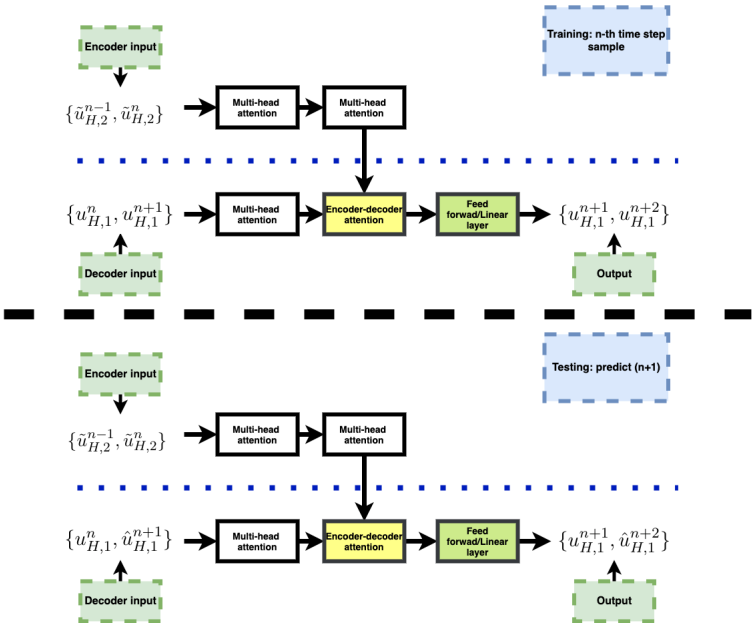


Figure 5: Data assimilation model training (top) and testing (bottom). The inputs of the encoder are $\{\tilde{u}^{n-1}_{H,2}, \tilde{u}^{n}_{H,2}\}$ which are the solutions of the known equation (with $\tilde{\kappa}$ as the permeability). The machine model predicts $u^{n}_{H,1}$ which can be regarded as a coarse prediction; this solution is then corrected by $\tilde{u}^{n}_{H,2}$ which is known and given.

The advantages of our method (Transformer) are still significant for this application. Transformer can handle multiple time series which originate from different spaces. To be more specific, the model can encode multiple time series and obtain multiple reduced order models of $u_{H,1}$ which is from the solution space of the target κ and $\tilde{u}_{H,2}$ which is from the known $\tilde{\kappa}$. Moreover, the encoder-decoder attention block is able to learn the connection between two encoded time series; i.e., the model can learn the interplay between two reduced order models. Besides, the model can handle the cumulative error and avoid solution blowing up.

5 Numerical examples

In this section, we will present some numerical examples. We solve two equations: linear and nonlinear parabolic equations. For each equation, we will first present the original example (the model is given) and then discuss the coarse solution assimilation results (the model is unknown). To generate the training and testing data, we use the splitting scheme (4) for the linear equation and (3) for the nonlinear equation. ω is set to be 0.5 for all examples. For the machine learning hyper-parameters, we set $n_e = n_d = 2$, the model hence directly mimics the splitting scheme.

5.1 Linear example

In the next two sections, we consider the following linear parabolic problem

$$u_t = \nabla \cdot (\kappa \nabla u), x \in [0, T] \times \Omega, \tag{7}$$

$$\frac{\partial u}{\partial x} = 0, x \in \partial\Omega, \tag{8}$$

$$u(x, 0) = u_0, \tag{9}$$

where $\Omega = [0,1]^2$ and $T = 0.00002$; u_0 is the initial condition, κ is the permeability field, they are demonstrated in Figure (6) and Figure (7).

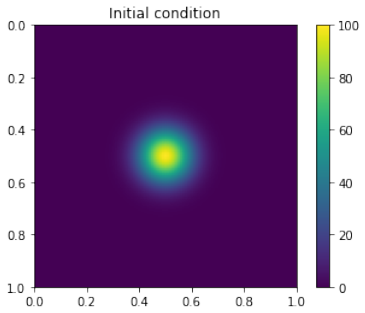


Figure 6: Initial condition used in the linear parabolic problems.

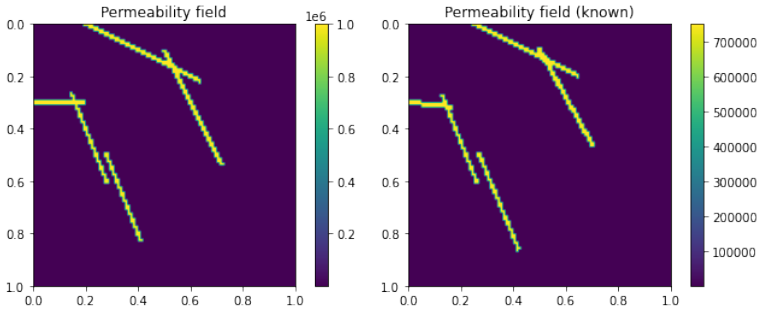


Figure 7: Left: the target permeability. In section (5.1) (linear) and (5.3) (nonlinear), this permeability is known; however, we assume it is unknown in the coarse solution assimilation examples. Right: the known permeability $\tilde{\kappa}$ which is closed to the target unknown permeability κ used in the coarse solution assimilation examples. Since $\tilde{\kappa}(x)$ is given, solution $\{\tilde{u}^n_{H,2}\}_{n=1}^N$ can be derived accordingly by the splitting scheme.

The time step $\tau = 0.00000002$ and we train the model with the first 25 steps and predict the remaining 974 up to the terminal time. We follow the first equation to design the scheme, i.e., $n_d = n_e = 2$. To measure the stability of the method, we calculate the L_2 relative error of $u^n_{H,1}$ and $u^n_{H,2}$ for all predicted time steps; the references are the solutions obtained by applying the splitting scheme directly. The errors are shown in Figure (8).

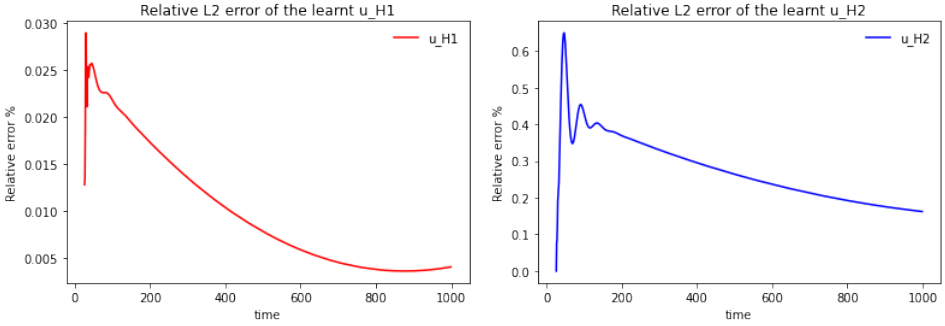


Figure 8: Example 1 (linear parabolic). Left, relative error of $u^n_{H,1}$ with respect to time steps. Right, relative error of $u^n_{H,2}$ with respect to time steps. The reference solutions are the solutions of the splitting scheme.

We also map the coarse solutions $u_H^n = u_{H,1}^n + u_{H,2}^n$ back to the fine scale and compute the L_2 relative error; the reference solution is the finite element solution on mesh 100×100 with the backward Euler scheme in time. The relative errors are shown in Figure (9).

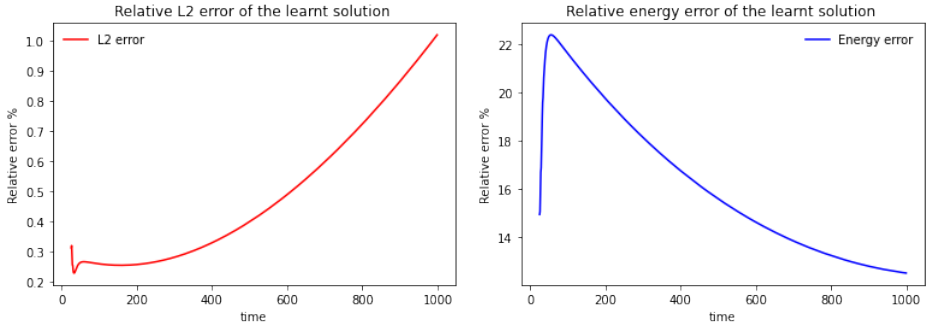


Figure 9: Example 1 (linear parabolic) results. Left, the L_2 relative error of the solution $(u_{H,1}^n + u_{H,2}^n)$ with respect to time steps. Right: the energy relative error of the solution. The reference solution is the FEM solution with mesh 100×100.

We can see from Figure (8), the model is able to handle the accumulative error and hence the prediction $(u_{H,1}^n)$ of the model is stable. Besides, the error of the prediction is also very small (less than 0.03%); $u_{H,2}^n$ is computed basing on the predictions $u_{H,1}^n$ of the model, it is also stable and accurate with relative error less than 0.7%. In Figure (9), we notice that the relative L_2 and energy errors of the solution (the fine mesh solution of the our proposed coarse solution u_H^n) are also very small and controllable. Similar results are observed in all following examples and our numerical results are very robust to the hyper-parameters and training, we hence conclude that our proposed method is accurate and stable.

5.2 Coarse solution assimilation for the linear example

To demonstrate our idea, we will solve the same linear equation (12). We will train the model with the first 60 steps and predict the remaining 939 steps; all the other settings (eg, source, time step...) are the same as before. We calculate L_2 relative error of $u_{H,1}^n$ for all predicted time steps; please note that there is no need to compute $u_{H,2}^n$ since the $u_{H,1}^n$ is corrected by the given $\tilde{u}_{H,2}^n$ instead. The result is presented in Figure (10).

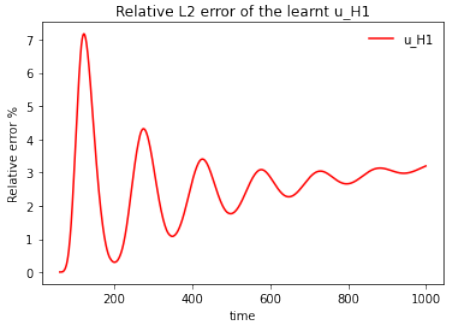


Figure 10: Results of the linear coarse solution assimilation example. Relative error of $u_{H,1}^n$ with respect to time steps.

We also map the coarse solutions $u_{H,1}^N$ and $u_H^N = u_{H,1}^N + \tilde{u}_{H,2}^N$ back to the fine scale and compute

the L_2 and energy relative errors. The idea is to correct $u_{H,1}^N$ with $\tilde{u}_{H,2}^N$ which is the correction of the approximated permeability. The results are shown in Figure (11) and please check Table (1) for the details.

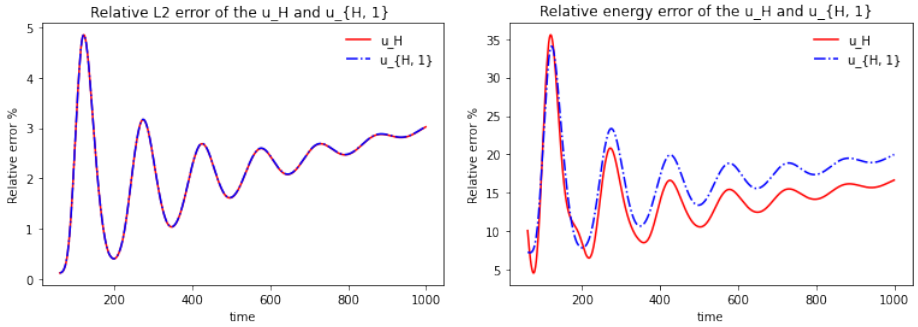


Figure 11: Results of the linear coarse solution assimilation example. Left: relative L_2 error with respect to time steps. Right: relative energy error with respect to time steps. Red color dashed line is the fine scale u_H^n (which maps the coarse solution to the fine scale mesh); while the blue curve is the fine scale $u_{H,1}^n$.

Coarse	L_2	Energy
$u_{H,1}$	2.262591%	17.142532%
u_H	2.25732%	14.435881%

Table 1: Linear coarse solution assimilation example relative errors. We here present the L_2 and the energy relative errors of the solutions obtained by mapping $u_{H,1}^n$ and u_H^n to the fine scale mesh. The errors are calculated as the average over all time steps.

We can observe from Figure (10) and Figure (11) both prediction and the solution are stable and are accurate. Since the target κ is not given, the model is supposed to learn $u_{H,1}$ basing on $\tilde{u}_{H,2}$, moreover, $\tilde{u}_{H,2}$ should correct and improve $u_{H,1}$. This can be seen from Table (1). The L_2 and energy errors are small; and adding $\tilde{u}_{H,2}$ indeed improves the relative energy error of the solution.

We want to give a comment before finishing the linear coarse solution assimilation example. It may be costly to pre-compute all $\tilde{u}_{H,2}^n$. We provide two strategies which can reduce the computational cost. Firstly, one can calculate $\tilde{u}_{H,2}^n$ by following the scheme (6). Compared to pre-compute all $\tilde{u}_{H,2}^n$, one avoids the computation of $\tilde{u}_{H,1}^n$ which is the contrast dependent solution of the approximation model $\tilde{\kappa}$. Secondly, we can train a model that predicts $\tilde{u}_{H,1}^n$ and calculate $\tilde{u}_{H,2}^n$ of the approximated model $\tilde{\kappa}$, this is exactly what we did in the section (5.1). Since the maximal relative error of $u_{H,2}^n$ is less than 0.6% in the last example (5.1), we believe our method can provide an accurate $\tilde{u}_{H,2}^n$ for later use in the assimilation. In all, this strategy consists of two stages. In the first stage, one can train a model for $\tilde{\kappa}$ and then predict and compute all $\tilde{u}_{H,1}^n$ and $\tilde{u}_{H,2}^n$. In the second stage, one can train a different model which relies on the pre-computed $\tilde{u}_{H,2}^n$ to perform the coarse data assimilation.

5.3 Nonlinear example

In the next two sections, we consider the following nonlinear parabolic problem:

$$u_t = \nabla \cdot (e^u \kappa_0(x) \nabla u), x \in [0, T] \times \Omega, \tag{10}$$

$$\frac{\partial u}{\partial x} = 0, x \in \partial\Omega, \tag{11}$$

$$u(x, 0) = u_0, \tag{12}$$

where $\Omega = [0, 1]^2$ and $T = 0.00002$; $u_0 = e^{(-(x_1-0.5)^2-(x_2-0.5)^2)/0.01}$ is the initial condition, κ_0 is the permeability field, they are demonstrated in Figure (6) and Figure (7). We set $\tau = 0.00000002$ and we train the model with the first 60 time steps and predict the remaining 939 steps. We calculate the L_2 relative error of $u^n_{H,1}$ and $u^n_{H,2}$ for all predicted time steps; the references are the solutions obtained by applying the splitting scheme directly. The errors are shown in Figure (12).

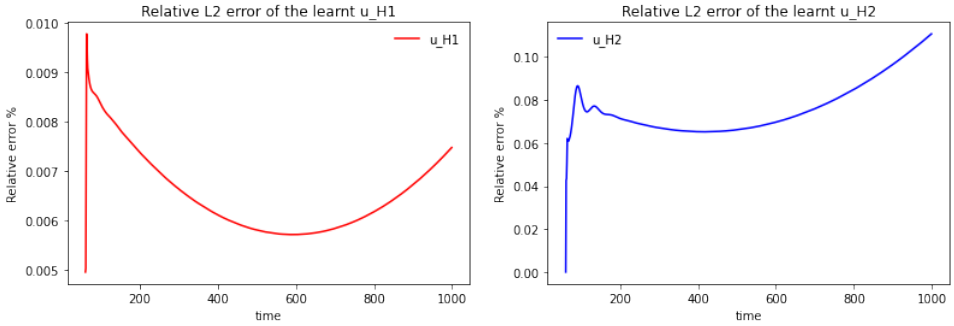


Figure 12: Results of the nonlinear example. Left: the L_2 relative error of $u^n_{H,1}$ with respect to time steps. Right: the L_2 relative error of the solution $u^n_{H,2}$ with respect to time steps.

We also map the coarse solutions $u^n_H = u^n_{H,1} + u^n_{H,2}$ back to the fine scale and compute the L_2 and energy relative error; the reference solution is the finite element solution on mesh 100×100 with the backward Euler scheme in time. The relative errors are shown in Figure (13).

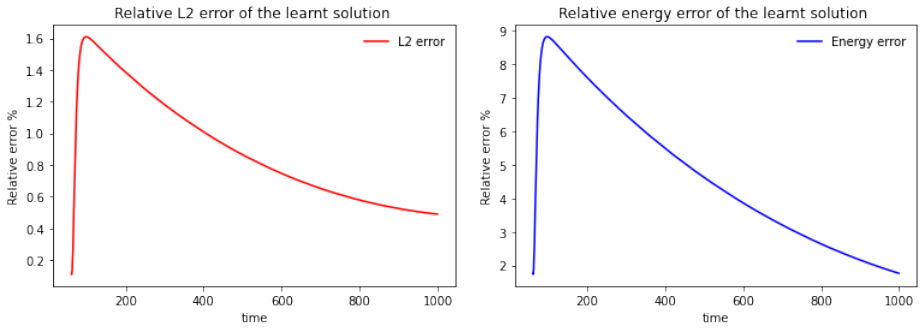


Figure 13: Example 3 (nonlinear parabolic problem) results. Left, the L_2 relative error of the solution $(u^n_{H,1} + u^n_{H,2})$ with respect to time steps. Right: the energy relative error of the solution. The reference solution is the FEM solution with mesh 100×100.

Similar with the linear example, the nonlinear model is also accurate and stable (please check Figure (12) left): the relative error in the prediction of $u_{H,1}$ is less than 0.01%; the computed $u_{H,2}$ which relies on the prediction of the model is also small (less than 0.15%). If we look at

Figure (13), the fine scale solution which is obtained by mapping $u^n_{H,1} + u^n_{H,2}$ back to the fine scale mesh is also stable; the relative errors also show that our method is very accurate.

5.4 Coarse solution assimilation for the nonlinear example

To demonstrate our idea, we will solve the same nonlinear equation (12). We will train the model with the first 50 steps and predicts the remaining 949 steps; all the other settings (eg, source, time step...) are the same as before. We calculate L_2 relative error of $u^n_{H,1}$ for all predicted time steps; please note that there is no need to compute $u^n_{H,2}$ since the $u^n_{H,1}$ is corrected by the given $\tilde{u}^n_{H,2}$. The result is presented in Figure (14).

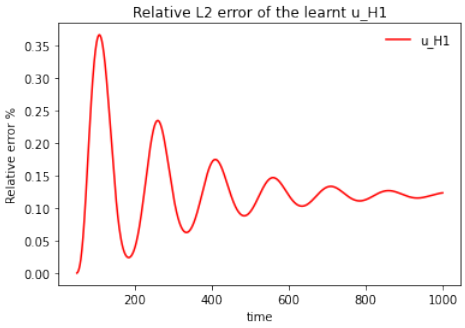


Figure 14: Results of the nonlinear coarse solution assimilation example. The L_2 relative error of $u^n_{H,1}$ with respect to time steps.

We also map the coarse solutions $u^n_{H,1}$ and $u^n_H = u^n_{H,1} + \tilde{u}^n_{H,2}$ back to the fine scale and compute the L_2 and energy relative errors. The idea is to correct $u^n_{H,1}$ with $\tilde{u}^n_{H,2}$ which is the correction of the approximated permeability. The results are shown in Figure (15) and please check Table (2) for the details.

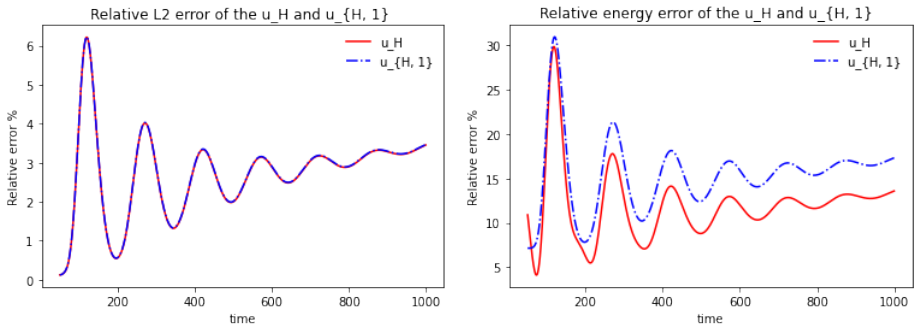


Figure 15: Results of the nonlinear coarse solution assimilation example. Left: relative L_2 error with respect to time steps. Right: relative energy error with respect to time steps. Red color dashed line is the fine scale u_H; while the blue curve is the fine scale $u_{H,1}$.

Coarse	L_2	Energy
$u_{H,1}$	2.742292%	15.498801%
u_H	2.731022%	11.996128%

Table 2: Nonlinear coarse solution assimilation example relative errors. We here present the L_2 and the energy relative errors of the solutions obtained by mapping $u_{H,1}$ and u_H to the fine scale mesh. The errors are calculated as the average over all time steps.

Figure (14) demonstrates the error of the model prediction. The relative error is less than 0.4% and is very stable. The errors are shown in Figure (15), we can see that the solution is very stable and accurate. If we look at Table (2), the $\tilde{u}_{H,2}$ indeed corrects $u_{H,1}$ and improves the solution. To reduce the cost of precomputing $\tilde{u}^n_{H,2}$, one can use the same strategies provided in section (5.2). In particular for the strategy two which trains a separate model for $\tilde{u}^n_{H,2}$, the maximal relative error of $u^n_{H,2}$ in example 5.3 is less than 0.1%, we believe that the trained model can provide an accurate $u^n_{H,2}$ for the future use in the coarse solution assimilation.

6 Conclusion

In this work, we design machine learning approaches to accelerate the contrast-independent splitting scheme and solve coarse data assimilation problems. We proposed three strategies, i.e., one can either learn u_1 which represents the main feature of the problem, or u_2 which is the contrast independent part of the solution, or both u_1 and u_2. We focus on learning u_1 because the computation of u_1 is implicit and time consuming if we follow the standard splitting scheme. We interpret the splitting scheme as a language translation problem and hence use the Transformer to solve it. Transformer is powerful in our problem since it can encode the inputs of different spaces and reduce the accumulative error. We also find that the splitting is a great platform to solve the coarse solution assimilation problem. All three strategies can be used and our machine learning model is also powerful in solving the coarse solution assimilation problems. We demonstrate and verify our proposed method with four examples, and the results show that our method is stable and accurate. In the future, we will study the other two strategies and improve our model.

7 Acknowledgement

Guang Lin gratefully acknowledges the support of the National Science Foundation (DMS-1555072, DMS-1736364, DMS-2053746, and DMS-2134209), and Brookhaven National Laboratory Subcontract 382247, and U.S. Department of Energy (DOE) Office of Science Advanced Scientific Computing Research program DE-SC0021142).

References

[1] B. Chetverushkin, E. Chung, Y. Efendiev, S.-M. Pun, and Z. Zhang. Computational multiscale methods for quasi-gas dynamic equations. *Journal of Computational Physics*, 440:110352, 2021.

[2] E. Chung, Y. Efendiev, W. T. Leung, S.-M. Pun, and Z. Zhang. Multi-agent reinforcement learning accelerated mcmc on multiscale inversion problem. *arXiv preprint arXiv:2011.08954*, 2020.

[3] E. Chung, Y. Efendiev, S.-M. Pun, and Z. Zhang. Computational multiscale methods for parabolic wave approximations in heterogeneous media. *arXiv preprint arXiv:2104.02283*, 2021.

[4] E. Chung, W. T. Leung, S.-M. Pun, and Z. Zhang. A multi-stage deep learning based algorithm for multiscale model reduction. *Journal of Computational and Applied Mathematics*, 394:113506, 2021.

[5] E. T. Chung, Y. Efendiev, and T. Hou. Adaptive multiscale model reduction with generalized multiscale finite element methods. *Journal of Computational Physics*, 320:69–95, 2016.

[6] E. T. Chung, Y. Efendiev, and C. Lee. Mixed generalized multiscale finite element methods and applications. *SIAM Multiscale Model. Simul.*, 13:338–366, 2014.

[7] E. T. Chung, Y. Efendiev, and W. T. Leung. Generalized multiscale finite element methods for wave propagation in heterogeneous media. *Multiscale Modeling & Simulation*, 12(4):1691–1721, 2014.

[8] E. T. Chung, Y. Efendiev, and W. T. Leung. Constraint energy minimizing generalized multiscale finite element method. *Computer Methods in Applied Mechanics and Engineering*, 339:298–319, 2018.

[9] E. T. Chung, Y. Efendiev, and W. T. Leung. Constraint energy minimizing generalized multiscale finite element method in the mixed formulation. *Computational Geosciences*, 22(3):677–693, 2018.

[10] E. T. Chung, Y. Efendiev, and W. T. Leung. Fast online generalized multiscale finite element method using constraint energy minimization. *Journal of Computational Physics*, 355:450–463, 2018.

[11] E. T. Chung, Y. Efendiev, W. T. Leung, and W. Li. Contrast-independent partially explicit time discretizations for nonlinear multiscale problems. 2021.

[12] E. T. Chung, Y. Efendiev, W. T. Leung, and P. N. Vabishchevich. Contrast-independent partially explicit time discretizations for multiscale wave problems. *arXiv preprint arXiv:2102.13198*, 2021.

[13] E. T. Chung, Y. Efendiev, W. T. Leung, M. Vasilyeva, and Y. Wang. Non-local multi-continua upscaling for flows in heterogeneous fractured media. *Journal of Computational Physics*, 372:22–34, 2018.

[14] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

[15] W. E and B. Engquist. Heterogeneous multiscale methods. *Comm. Math. Sci.*, 1(1):87–132, 2003.

[16] Y. Efendiev, J. Galvis, and T. Hou. Generalized multiscale finite element methods (GMsFEM). *Journal of Computational Physics*, 251:116–135, 2013.

[17] Y. Efendiev and T. Hou. *Multiscale Finite Element Methods: Theory and Applications*, volume 4 of *Surveys and Tutorials in the Applied Mathematical Sciences*. Springer, New York, 2009.

[18] P. Henning, A. Målqvist, and D. Peterseim. A localized orthogonal decomposition method for semi-linear elliptic problems. *ESAIM: Mathematical Modelling and Numerical Analysis*, 48(5):1331–1349, 2014.

[19] T. Hou and X. Wu. A multiscale finite element method for elliptic problems in composite materials and porous media. *J. Comput. Phys.*, 134:169–189, 1997.

[20] P. Jenny, S. Lee, and H. Tchelepi. Multi-scale finite volume method for elliptic problems in subsurface flow simulation. *J. Comput. Phys.*, 187:47–67, 2003.

[21] A. Katrutsa, T. Daulbaev, and I. Oseledets. Black-box learning of multigrid parameters. *Journal of Computational and Applied Mathematics*, 368:112524, 2020.

[22] W. T. Leung, G. Lin, and Z. Zhang. Nh-pinn: Neural homogenization based physics-informed neural network for multiscale problems. *arXiv preprint arXiv:2108.12942*, 2021.

[23] L. Liu, T. Zeng, and Z. Zhang. A deep neural network approach on solving the linear transport model under diffusive scaling. *arXiv preprint arXiv:2102.12408*, 2021.

[24] G. Pang, M. D'Elia, M. Parks, and G. E. Karniadakis. npinns: nonlocal physics-informed neural networks for a parametrized nonlocal universal laplacian operator. algorithms and applications. *Journal of Computational Physics*, 422:109760, 2020.

[25] A. Roberts and I. Kevrekidis. General tooth boundary conditions for equation free modeling. *SIAM J. Sci. Comput.*, 29(4):1495–1510, 2007.

[26] G. Samaey, I. Kevrekidis, and D. Roose. Patch dynamics with buffers for homogenization problems. *J. Comput. Phys.*, 213(1):264–287, 2006.

[27] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[28] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[29] Y. Wang, S. W. Cheung, E. T. Chung, Y. Efendiev, and M. Wang. Deep multiscale model learning. *Journal of Computational Physics*, 406:109071, 2020.

[30] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena. Self-attention generative adversarial networks. In *International conference on machine learning*, pages 7354–7363. PMLR, 2019.

[31] Z. Zhang, E. T. Chung, Y. Efendiev, and W. T. Leung. Learning algorithms for coarsening uncertainty space and applications to multiscale simulations. *Mathematics*, 8(5):720, 2020.