FLUID-GPT (Fast Learning to Understand and Investigate Dynamics with a Generative Pre-Trained Transformer): Efficient Predictions of Particle Trajectories and Erosion

Steve D. Yang,[a,‖] *Zulfikhar A. Ali,*[b,‖] *and Bryan M. Wong*[a,b*]

a) Materials Science & Engineering Program, University of California-Riverside, Riverside, CA 92521, USA

b) Department of Chemistry and Department of Physics & Astronomy, University of California-Riverside, Riverside, CA 92521, USA

‖S.D.Y. and Z.A.A. contributed equally to this work

*E-mail: bryan.wong@ucr.edu, Website: http://www.bmwong-group.com

*Corresponding author. E-mail: bryan.wong@ucr.edu. Webpage: http://www.bmwong-group.com

Abstract

The deleterious impact of erosion due to high-velocity particle impingement adversely affects a variety of engineering/industrial systems, resulting in irreversible mechanical wear of materials/components. Brute force computational fluid dynamics (CFD) calculations are commonly used to predict surface erosion by directly solving the Navier Stokes equations for the fluid and particle dynamics; however, these numerical approaches often require significant computational resources. In contrast, recent data-driven approaches using machine learning (ML) have shown immense promise for more efficient and accurate predictions to sidestep the computationally demanding CFD calculations. To this end, we have developed FLUID-GPT (Fast Learning to Understand and Investigate Dynamics with a Generative Pre-Trained Transformer), a new hybrid ML architecture for accurately predicting particle trajectories and erosion on an industrial-scale steam header geometry. Our FLUID-GPT approach utilizes a Generative Pre-Trained Transformer 2 (GPT-2) with a Convolutional Neural Network (CNN) for the first time to predict surface erosion using only information from five initial conditions: particle size, main-inlet speed, main-inlet pressure, sub-inlet speed, and sub-inlet pressure. Compared to the Long- and Short-Term Memory (LSTM) ML techniques used in previous work, our FLUID-GPT model is much more accurate (a 54% decrease in mean squared error) and efficient (70% less training time). Our work demonstrates that FLUID-GPT is an accurate and efficient ML approach for predicting time-series trajectories and their subsequent spatial erosion patterns in these complex dynamic systems.

1. Introduction

Erosion due to high-velocity particle impingement continues to be a topic of pressing concern due to its deleterious effects in a variety of energy and technological industries. For example, erosion processes result in irreversible mechanical wear of materials/components in petroleum refining,[1] aircraft rotor/engine blades,[2,3] and pipelines in coal-fired power plants.[4,5] Recent studies have estimated that the financial loss due to erosion can reach up to several billions of US dollars in industrialized nations.[6] To mitigate these effects, computational fluid dynamics (CFD) is commonly used to solve the Navier-Stokes equations for the fluid and particle dynamics to shed insight into the specific mechanisms involved in these erosion processes. However, CFD calculations often require significant computational time and high-performance computing hardware, particularly for large-scale structures used in industrial power plants.

In recent years, there has been a rapid paradigm shift from “traditional” computational approaches to data-driven science, largely due to advances in computational power and the increasing availability of data. In the context of CFD and erosion calculations, recent machine learning (ML) approaches have shown remarkable accuracy and efficiency in a variety of systems.[7–18] In particular, previous work by us utilized a hybrid Long- and Short-Term Memory (LSTM) with a 3-dimensional Convolutional Neural Network (CNN) to predict particle trajectories and surface erosion, respectfully, in an industrial-scale boiler header.[18] While our approach was able to successfully predict surface erosion using only five initial conditions as input, the LSTM training was computationally expensive due to its recurrence-based architecture, taking approximately 26 hours on 32 parallel CPUs.

To overcome this computational bottleneck, we present a new hybrid ML method, codenamed FLUID-GPT: Fast Learning to Understand and Investigate Dynamics with a Generative Pre-Trained Transformer. FLUID-GPT utilizes a Generative Pre-Trained Transformer 2 (GPT-2) with a 3D CNN. The primary objective of FLUID-GPT revolves around the accurate prediction of particle trajectories and erosion phenomena within a sprawling industrial-scale boiler header.[21] GPT-2 is an attention-based model that falls under the umbrella of transformer models. Initially developed for tasks like natural language processing (NLP) and translation,[19] it has recently gained widespread attention due to its utilization in interactive Artificial Intelligence (AI) chatbots and forecasting models for time-series data.[20–22] A transformer model employs an encoder-decoder architecture in which the encoder generates a representation capturing relevant information from the input data. Subsequently, the decoder utilizes this representation to create output sequences through the attention mechanism iteratively. Recent research has demonstrated the effectiveness of focusing exclusively on the decoder for language modeling, leading to the development of a GPT model.[23]

This study employs our FLUID-GPT ML approach to forecast particle trajectories based on five initial parameters: particle size, main-inlet speed, main-inlet pressure, sub-inlet speed, and sub-inlet pressure. Subsequently, erosion predictions are generated using a CNN-based approach, utilizing the trajectories produced by our time-series models. We comprehensively describe our algorithms and conduct a comparative analysis of training efficiency and accuracy between our FLUID-GPT approach, LSTM, and Bidirectional LSTM (BiLSTM) for predicting particle trajectories and surface erosion. Finally, we offer a concise summary and discuss potential future applications of our hybrid ML approach. In conclusion, our study underscores the efficiency and accuracy of FLUID-GPT in analyzing sequential data, marking a significant step forward in predicting intricate particle dynamics.

2. Computational Methods

Our FLUID-GPT hybrid machine learning model combines GPT-2 and CNN within the PyTorch framework. We customized the GPT-2 architecture from the Transformer PhysX (TrphysX) package[21] to train our CFD dataset, which we carried out on a single Nvidia K80 GPU. The following sections describe the GPT-2 and CNN architecture, our data collection, and transformation process. We then describe our model training approach, including an early stopping criterion, which we used to compare computational efficiency across the various machine learning algorithms.

2.1 GPT-2 Model Architecture

Our utilization of the GPT-2 model for time-series analysis stems from its inherent ability to discern patterns and interconnections within sequential data points autonomously. It is important to highlight that the original GPT-1 algorithm was primarily designed for pre-training and focusing on acquiring linguistic structure and grammar. Subsequently, this foundational model undergoes refinement through supervised fine-tuning, requiring substantial labeled data pertinent to the specific task.[24] However, this reliance on supervised fine-tuning can present limitations as it necessitates the acquisition and processing of considerable labeled datasets, potentially restraining the model's versatility.

The more recent GPT-2 model, distinguished by its augmented input capacity and model size compared to GPT-1, introduced a groundbreaking approach known as "unsupervised fine-tuning." This innovation empowers the model to fine-tune for specialized tasks without requiring extensive task-specific data volumes. This flexibility enables the model to seamlessly adapt to varying tasks with reduced data prerequisites.[24] We leverage this unique feature by augmenting our input data to incorporate the five initial parameters (discussed in the Introduction) utilized in CFD calculations. This augmentation ensures the model's comprehensive learning of particle trajectories, preventing the need for supplementary training. Further elaboration on our feature engineering can be found in Section 2.3.1 - Data Collection.

GPT-2 model training is accomplished by calculating attention scores for each timestep, which measures the degree of association between the current timestep and other timesteps in the sequence.[23] In contrast to recurrent ML models, which process timesteps one at a time and propagate information from one stage to the next through a hidden state, the GPT-2 model employs self-attention. This enables the model to capture more complex relationships between the different timesteps in the sequence.

Positional encoding[23] is pivotal in enabling the GPT-2 model to distinguish between distinct timesteps accurately. This involves the introduction of unique patterns for each timestep, facilitating the model's ability to discern temporal variations. The GPT-2 model adeptly incorporates the foundational Transformer model's positional encoding approach. These patterns specify each timestep by integrating sine- and cosine-based positional patterns into the data's feature dimension. This can be equated to the addition of timestamps to a sequence of events, facilitating well-informed predictions and upholding chronological coherence throughout the prediction sequence.[23]

Figure 1 depicts the GPT-2 architecture,[24] composed of vertically stacked transformer decoder blocks integrated with positional encoding. Within each decoder block, a multi-head masked attention and a multi-layer perceptron (MLP) are enveloped by normalization[25] and dropout layers. The input time-series data is organized as an array (samples, timesteps, features) in the multi-head masked attention, where each timestep corresponds to a distinct set of features. This array's feature dimension is partitioned and allocated to individual attention heads, enabling simultaneous processing and comprehensive treatment of diverse aspects within the input sequence. This parallelization boosts the model's capability to capture short- and long-range dependencies, culminating in heightened predictive stability and precision.

As shown in the example in Figure 1, the input is distributed across four attention heads. Each attention head autonomously transforms the input sequence into three distinct vectors: query (Q), key (K), and value (V). These vectors are subsequently employed to calculate attention scores using the following formula:[23]

$$A(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \tag{1}$$

where A designates the attention score matrix, Q stands for the query vector, K^T denotes the transposed key vector, V represents the value vector, and d_k signifies the dimension of the Q and K vectors. The Q vector corresponds to the current timestep within the considered input sequence, while K encompasses insights about all timesteps within the input sequence. The dot product between the Q and K establishes the similarity or association between different timesteps, enabling the model to concentrate on relevant parts of the data. The V vector carries the data to which the model directs its attention and is used to generate the output. After the computation of attention scores, the individual attention heads merge via concatenation in the final linear layer, culminating in deriving the final attention matrix output.

Figure 2(a) illustrates the learning process involving time-series data. This data is segmented into portions of length **ω**, where **ω** is the window size. These segments undergo systematic processing utilizing a "striding" technique, wherein the window traverses the sequence with stride **s**, resulting in overlapping parts. In the training phase, the last data point within the window is omitted from the training input. In contrast, the initial data point is excluded from the training label – a visual representation of this is provided in Figure 2(b). This strategic arrangement entrusts the model to assimilate insights from neighboring segments and derive contextual understanding across the sequence.

Subsequently, the multi-head masked attention layer generates Q, K, and V vectors employing the training input data. These vectors then play a role in computing attention scores through Equation (1), producing the attention output depicted in Figure 2(c). These scores are used to predict subsequent timestep values. Training the GPT model involves the iterative refinement of neural network components responsible for generating appropriate Q, K, and V vectors. This process enables the model to learn and generate attention scores that contribute to accurate predictions of future values within the time series.

The input sequence of GPT-2 undergoes a twofold segmentation process. Firstly, the time steps are partitioned into windows, each encapsulating a localized context of consecutive timesteps. Following the temporal segmentation, the feature dimension is distributed across separate attention heads, facilitating parallelized processing throughout the sequence. This strategic fusion of windowing and attention head allocation endows the GPT architecture to effectively process and predict time-series data, capturing intricate sequential relationships comprehensively.

Figure 3 shows how the GPT-2 algorithm predicts sequential values. We only use the initial timesteps as input and the entire 50 timesteps as label data. The trained GPT-2 model iteratively predicts the next timesteps until it reaches the last (50th) step. Notably, GPT-2 is an autoregressive model[26,27] where each timestep is generated based on all the previous timesteps. For example, the nth timestep, t_n, is predicted based on t_{n-1}, t_{n-2}, ..., t_2, and t_1. This timestep prediction method has an advantage over LSTM, which predicts t_n based only on t_{n-1}. Furthermore, as described in our previous machine learning study, the "forget" gate in the LSTM algorithm can forget previous information, which could adversely affect the predictions.[18] For these reasons, GPT-2 possesses several advantages in accuracy and training efficiency since it simultaneously accepts all timesteps and calculates attention scores (via the dot product) to generate contextually appropriate predictions for the next time step.

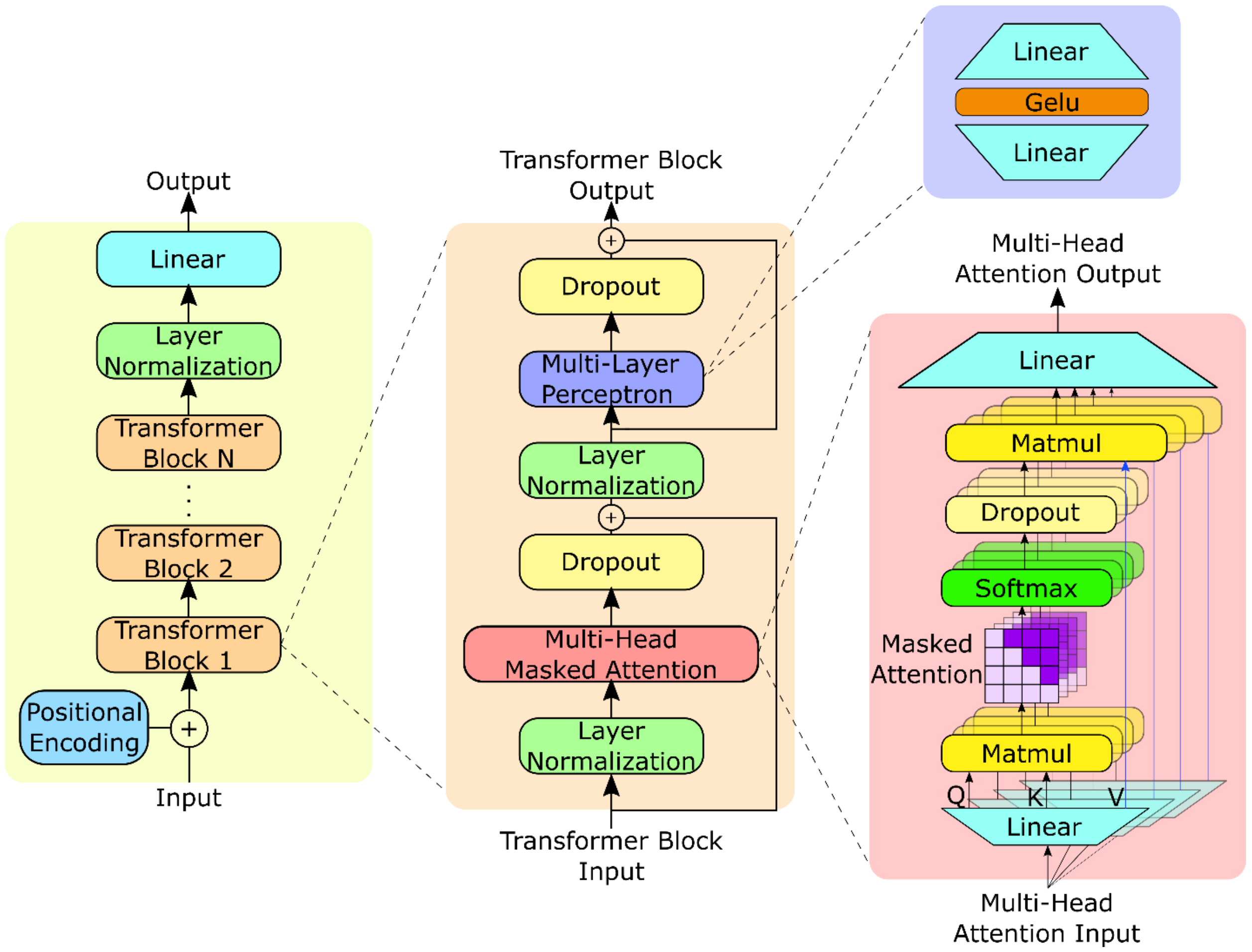


Figure 1. GPT-2 model architecture. The GPT-2 model contains N Transformer decoder blocks, as shown in the left panel. Each decoder block (center panel) includes a multi-head masked attention layer, a multi-layer perceptron layer, normalization, and dropout layers. The residual connection (branching line to the addition operator) allows the block to learn from the previous block's input. The multi-head masked attention layer (right panel) calculates attention scores using Q, K, and V vectors to capture sequential relationships in the input sequence.

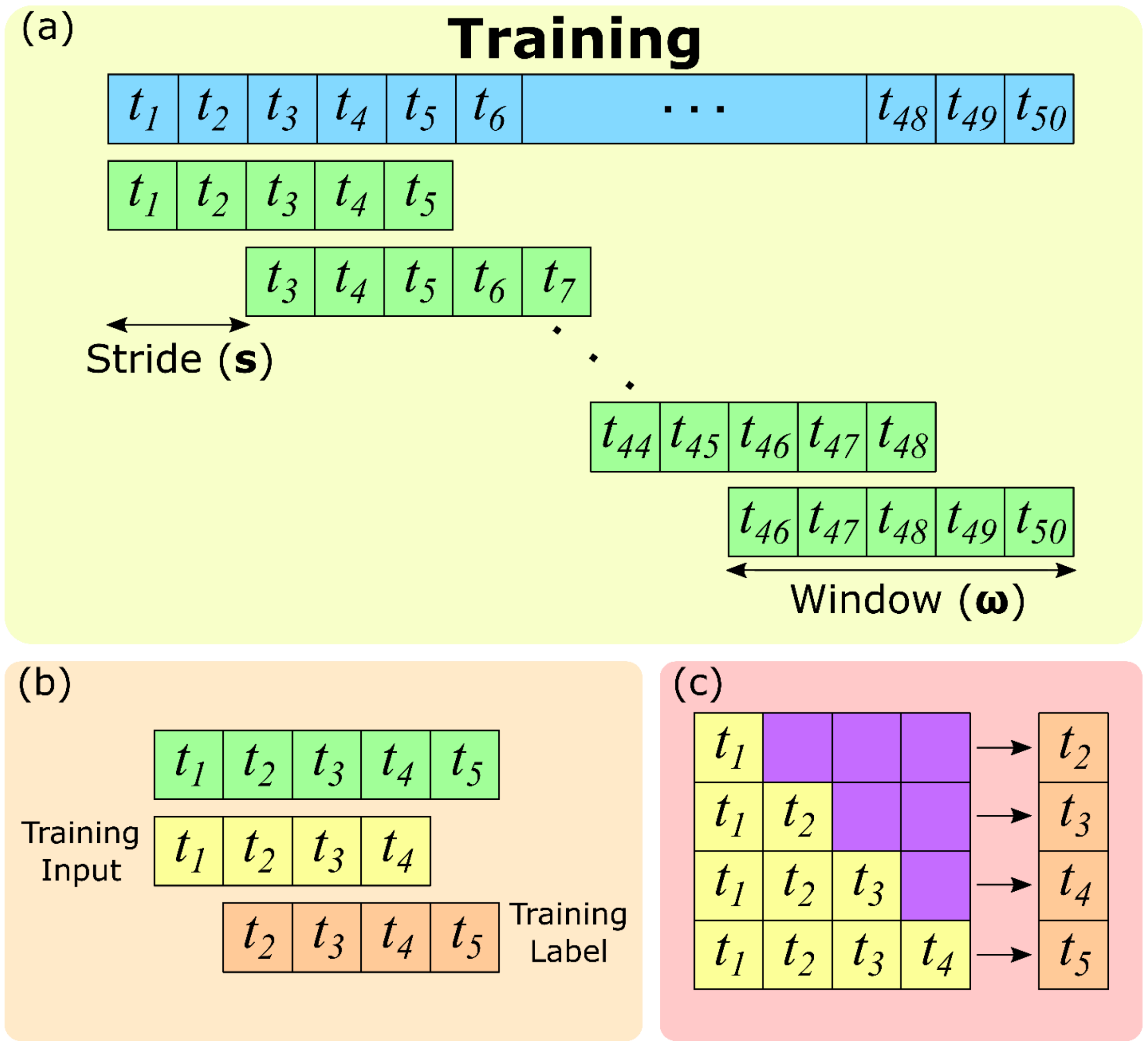

(a)
Training
t_1 t_2 t_3 t_4 t_5 t_6 ... t_{48} t_{49} t_{50}
t_1 t_2 t_3 t_4 t_5
t_3 t_4 t_5 t_6 t_7
Stride (**s**)
t_{44} t_{45} t_{46} t_{47} t_{48}
t_{46} t_{47} t_{48} t_{49} t_{50}
Window (**ω**)
(b)
t_1 t_2 t_3 t_4 t_5
Training Input
t_1 t_2 t_3 t_4
t_2 t_3 t_4 t_5
Training Label
(c)
t_1 → t_2
t_1 t_2 → t_3
t_1 t_2 t_3 → t_4
t_1 t_2 t_3 t_4 → t_5

Figure 2. GPT-2 training process. Panel (a) shows the division of the 50 timesteps into segments using window and stride parameters (**ω** = 5 and **s** = 2, respectively). Panel (b) displays each segment's input and labels data points during model training. Panel (c) demonstrates the computation of the attention layer's output using Equation 1, which involves masking certain positions in the input sequence to zero out their attention scores (shown as purple boxes).

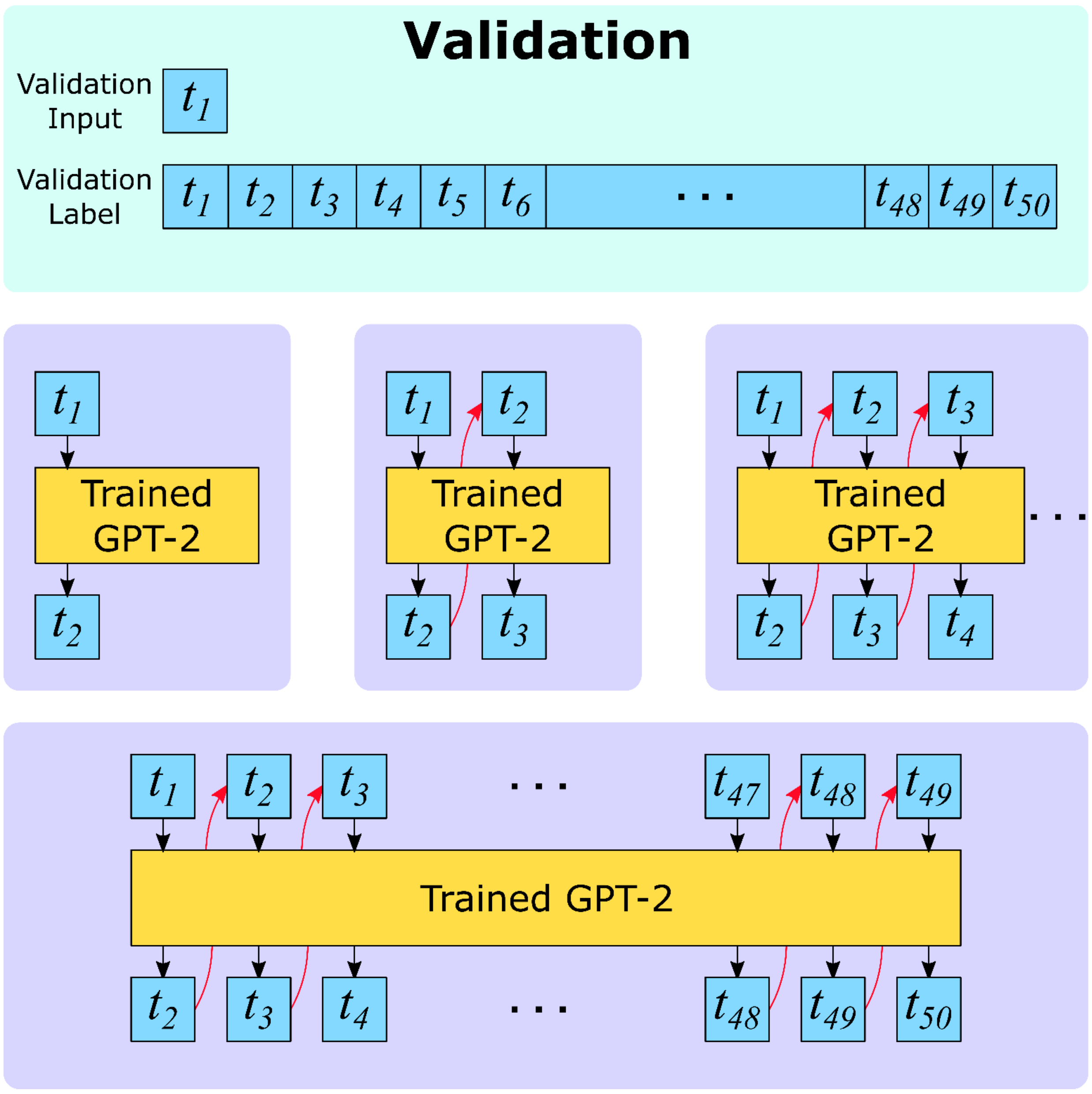


Figure 3. GPT-2 validation process. The trained GPT-2 model predicts all 50 timesteps based on the initial timestep, generating subsequent timesteps based on the previous ones until it reaches the last timestep of the labeled data.

2.2 CNN Architecture

The trajectories generated by the time-series models (GPT-2, BiLSTM, or LSTM) serve as input data for a 3D CNN model designed to predict the surface erosion rate. Utilizing the strengths of a 3D CNN (a type of advanced neural network) becomes crucial for our goals. This model is adept at recognizing patterns in 3D data, such as our time-series trajectories, making it better at predicting erosion. Enhancing its ability to understand complex spatial relationships contributes to its prediction accuracy.[28]

The 3D CNN utilizes specific filters known as kernels to analyze localized portions of input data and extract relevant information. These kernels, characterized by designated squared dimensions, filter the data and emphasize significant features while discarding unnecessary details. The process includes a stride, indicating the kernel's step size as it moves across the data. Additionally, max pooling is applied to further condense and preserve key spatial insights. As the process unfolds, gathered details are progressively integrated, enhancing the network's understanding of spatial relationships within the data. Introducing supplementary layers helps streamline the computation and maintain vital spatial information. The 3D CNN identifies and captures intrinsic data features through an iterative approach.

Our CNN model's success in predicting erosion is due to its ability to recognize spatial patterns that greatly affect erosion in the boiler header. Specifically, CNN identifies areas where particles encounter the surface because of fluid movement, which matches places with higher erosion rates. 3D convolutional layers allow the model to utilize initial conditions and how particles are positioned, revealing important connections between these factors and erosion values. These abilities make the CNN model a good fit for predicting erosion in fluid flow systems.

2.3 Data Collection, Model Training, and Early Stopping

2.3.1 Data Collection

We obtained particle trajectories and erosion data by running simulations in ANSYS Fluent 19.2 using the geometry of an OP-650 boiler header.[29–31] The obtained erosion dataset from our CFD simulations captures erosion observations at the simulations' final time. The particle trajectory data extracted from the CFD simulations constitutes a time-series dataset spanning 50 timesteps with a 2.49 ms time interval. To investigate the effects of various initial conditions, we adjusted the particle size, main-inlet speed, main-inlet pressure, sub-inlet speed, and sub-inlet pressure. Our previous study[18] provides further details and background on the CFD equations, configurations, and initial condition settings, which we also used for this work. In short, ANSYS Fluent calculates erosion by integrating force balance equations[32] over the particle trajectories, which we expanded upon in two stages. First, we utilized a time-series machine learning architecture to forecast particle trajectories based on initial conditions. Subsequently, we leveraged a 3D convolutional machine learning model that processes the predicted trajectories to predict surface erosion profiles.

Our dataset encompasses 3125 samples, each comprising *x*, *y*, and *z* coordinates of 196 particles across 50 timesteps. The 196-particle order from the CFD data was shuffled to minimize the data order dependency. Accounting for fluid flow fluctuations, we augmented each particle's trajectory with five initial condition parameters for every timestep. This yielded a dataset with $3125 \times 50 \times 196 \times 8$ dimensions. The last two dimensions were merged to obtain a final trajectory dataset with a shape of $3125 \times 50 \times 1568$, signifying the number of samples, time steps, and features, respectively.

Furthermore, an erosion dataset (3125 × 38312) was crafted, characterizing erosion values across the surface mesh of the boiler header. We shuffled the 3125 samples and divided them into training, validation, and test sets. For this division, the test set comprised 10% of the overall data, while the remainder was split between the training and validation sets in an 8:2 ratio. To avoid any risk of overfitting, we applied K-fold cross-validation[33,34] with $k = 5$. This process yielded training, validation, and test sets containing 2250, 562, and 313 samples, respectively.

2.3.2 Model Training

The FLUID-GPT approach developed in this study is a hybrid machine-learning model to predict particle trajectories and surface erosion rates in the OP-650 boiler header. Specifically, we used the predicted trajectories from the time-series models (GPT-2, BiLSTM, or LSTM) as input data to a CNN model to predict the surface erosion rate. It is worth emphasizing that our FLUID-GPT approach can predict surface erosion rates *using only information from five initial conditions*: particle size, main-inlet speed, main-inlet pressure, sub-inlet speed, and sub-inlet pressure.

We fine-tuned the hyperparameters of three time-series models: GPT-2, LSTM, and BiLSTM, employing the CFD dataset. BiLSTM served as a reference to assess the sequential memory dependency and accuracy effects of LSTM.[35,36] Our GPT-2 model comprises 120,558,816 parameters, while LSTM and BiLSTM each entail 305,024,608 parameters. We adopted 4 LSTM and BiLSTM layers, observing that reduced layers led to inadequate prediction performance. Table S1 in the Supporting Information presents results on model parameter count, R^2 scores, and mean squared errors (MSEs) for the other LSTM and BiLSTM layers.

Hyperparameters for the GPT-2 model included the number of decoder blocks, attention heads, and window/stride dimensions. Our GPT-2 model employed an MLP with the Gaussian

Error Linear Unit (GELU) activation function. We chose this particular configuration due to its superior performance over the Rectified Linear Unit (ReLU) in transformer models for NLP tasks. GELU effectively mitigates the vanishing gradient problem and exhibits a smoothness property, contributing to improved performance.[37,38]

We optimized learning rates utilizing the CyclicLR (CLR) scheduler,[39] which progressively reduces the periodic peak's magnitude as training advances. CLR has shown superior performance over alternative learning rate schedules, such as step and exponential decay, resulting in accelerated convergence and improved accuracy.[39,40] The Adam optimization method was used for each model training.

The CNN processes the predicted trajectory from either GPT-2 or BiLSTM, along with five initial parameters, to forecast surface erosion rates. Our method incorporates four layers of 3D convolution and max pooling, employing the GELU activation function. We optimize CNN performance through systematic enhancements involving kernel size (2, 3), stride (1, 2) variations in convolutional layers, and kernel sizes (2, 3) in max pooling. Additionally, we explore different filter combinations for each set of four convolution layers, denoted as (2-4-8-16), (4-8-16-32), (8-16-32-32), and (10-20-30-40). This comprehensive optimization approach ensures a thorough exploration of hyperparameters, contributing to the improved effectiveness of our CNN model. We employed this optimization strategy for two time-series models: FLUID-GPT (GPT-2+CNN) and BiLSTM+CNN. Performance evaluation encompasses key metrics, including each algorithm's MSE, R^2 score, and training time.

2.3.3 Early Stopping

While training our GPT-2 and LSTM models, we employed convergence criteria to prevent overfitting and optimize computational efficiency. Specifically, we utilized an early stopping method, where the training process was stopped when the MSE fell below a predetermined threshold for three consecutive epochs. We set the threshold value at 0.0065 based on the results of the CLR scheduler learning rate optimization procedure. We applied the same threshold value and counter limits to ensure that both models were trained using similar convergence criteria for a fair comparison of their respective performances.

3. Results and Discussion

3.1 GPT and BiLSTM Hyperparameter Optimization

3.1.1 Learning Rates and Schedulers

We optimized the learning rate for the GPT-2 architectures using the CLR scheduler, starting at a value of 1×10^{-3}. We varied the learning rate using an order of magnitude reduction strategy, where each new learning rate was reduced by a power of 10 to determine the optimal value for efficient and accurate convergence, as shown in Figure 4. Our simulations showed that a learning rate of 1×10^{-7} achieved the best performance, as shown in Figures 4(a) and (b). These results demonstrate the importance of optimizing the learning rate for optimal convergence in GPT-2 architectures.

The BiLSTM model was trained in a similar fashion, as shown in Figures 4(c) and (d). A learning rate of 1×10^{-4} showed the lowest MSE loss among our variations, with the most stable training and validation loss profiles. The LSTM model was also trained, and we also tested the linear and Cosine Annealing Warm Restarts schedulers,[41] which are provided in Figures S1 – S5 in the Supporting Information.

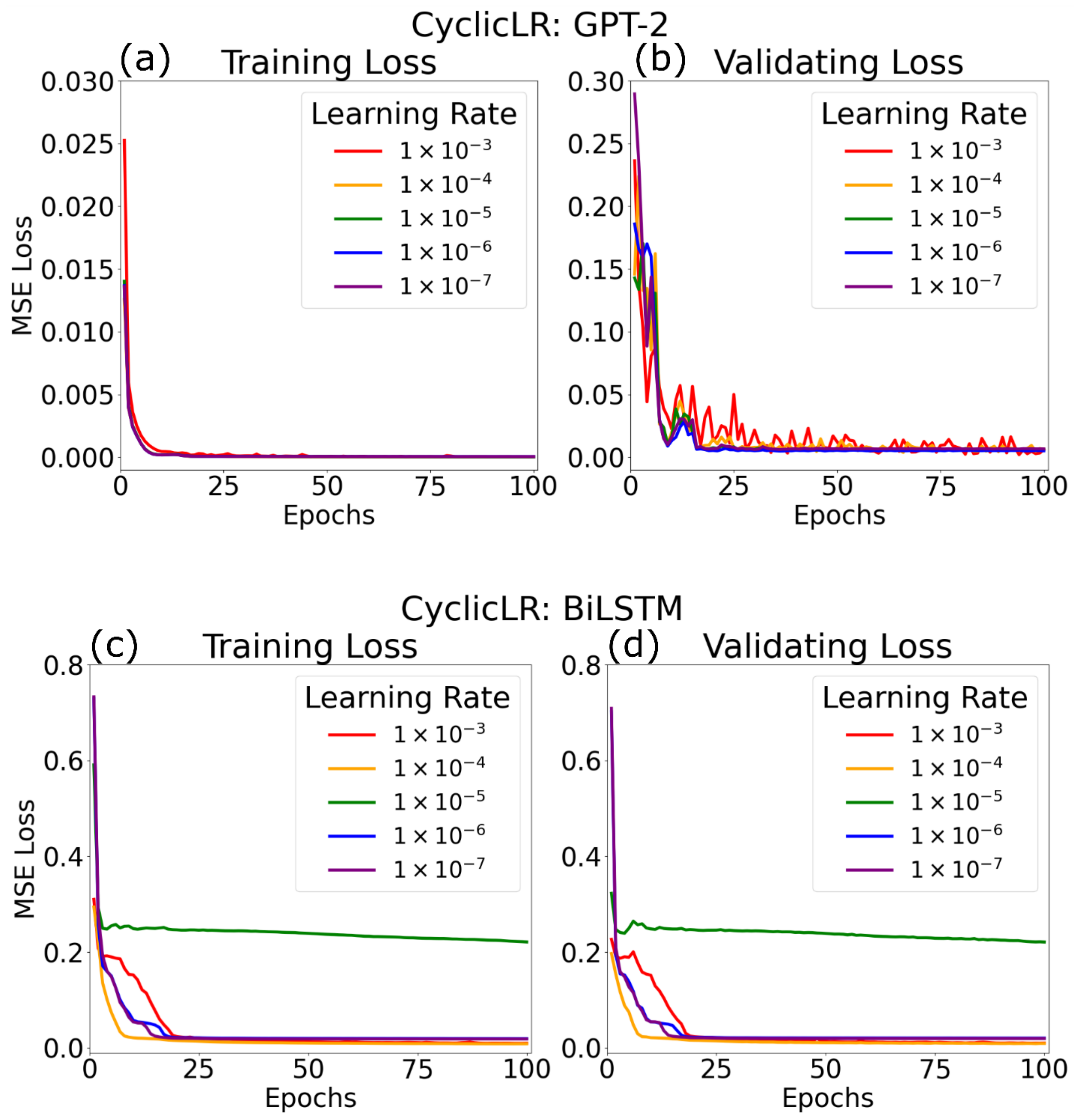


Figure 4. (a)-(b): Training and validation MSEs using the CLR scheduler for the GPT-2 architecture and (c)-(d) the BiLSTM model. The panels show the MSEs for different learning rates during training and validation.

3.1.2 Optimizing Window (ω) and Stride (s)

In the hyperparameter tuning process, we employed a grid search to determine the optimized values, with specific ranges, for each parameter: window (**ω** = 2, 4, 8, 16), stride (**s** = 2, 4, 8, 16), the number of decoder layers (2, 3, 4), and the number of attention heads (2, 4, 8, 16). Our results showed that 4 decoder block layers with 2 attention heads provided the most favorable outcome, striking a balance between the MSE and training duration. Figures 5(a) and (b) show the GPT-2 training duration and MSE loss for a range of **ω** and **s** pairs, respectively. Our observations unveiled a noteworthy trend: as the stride (**s**) decreased, the corresponding MSE loss exhibited a reduction, indicating an enhancement in prediction accuracy, especially when windows overlapped. However, it is essential to note that a decrease in the stride (**s**) also increased training duration. This effect can be attributed to generating more window steps, expanding the input data volume due to larger temporal overlaps.

We, therefore, sought to find a balance between accuracy and efficiency by selecting an appropriate value for **s**. We tested various values of **ω** and **s** and found that the early stopping criteria were triggered at different epochs for each combination of **ω** and **s**. Specifically, the early stopping was activated for 2/2, 4/2, 8/2, 16/2, and 8/4 (**ω**/**s**) pairs, resulting in a much shorter training duration for these five cases. In addition, we noticed that a **ω**:**s** ratio of 1:1 showed higher MSEs than other settings, as the windows did not overlap, and the relationship between the windows could not be learned. As shown in Figure 5, there is an optimal **ω** for each **s** that gives the lowest MSE. After carefully considering these factors, we chose **ω** = 4 and **s** = 2 as the best combination of parameters, resulting in the best balance between MSE loss and training duration compared to other settings.

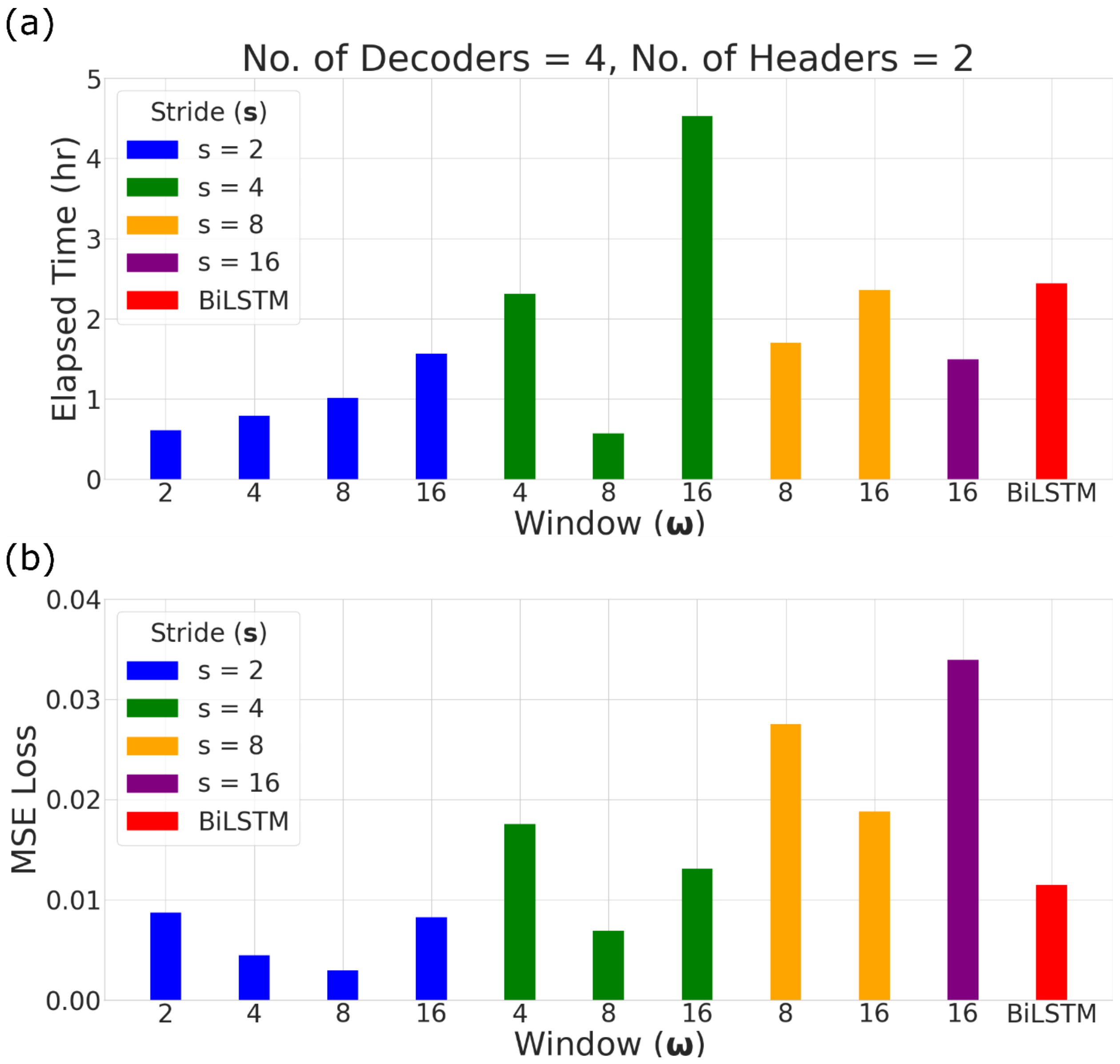


Figure 5. Window (**ω**) and stride (**s**) optimization for GPT-2 with a fixed number of decoder blocks and attention heads. Panel (a) displays the training duration, while panel (b) shows the validating MSE loss for different values of **ω** and **s** pairs. The colors indicate the value of **s** for each **ω**.

3.1.3 Number of Decoder Layers and Attention Heads

Our study extended to the influence of varying decoder block layers and attention heads. Training time accelerates as we reduce the number of decoders, albeit with a trade-off in increased MSE values. Figure 5 provides a general overview for configurations with 4 decoder blocks and 2 attention heads, whereas further details are given in Figures S6 – S11 within the Supporting Information. Smaller strides (**s** = 2 or 4) yield heightened performance, particularly with aligned attention head and stride values. These attributes modulate information blending and segment overlap. Smaller strides excel at finer-grained pattern capture, while larger strides reveal broader trends. Larger attention head numbers augment complex data handling and overall performance. The pivotal interplay between attention heads and stride is essential for orchestrating sequential information flow in input data, underpinning our analysis.

3.2 CNN Hyperparameter Optimization

We optimized the hyperparameters for two CNN models: FLUID-GPT (GPT-2+CNN) and BiLSTM+CNN. Due to the closely aligned results of these models, additional information on the optimization of BiLSTM+CNN is provided in Figures S12-S13 in the Supporting Information. Figure 6 illustrates the optimization of our FLUID-GPT model, utilizing a (8-16-32-32) convolution filter combination. Further details regarding the convolution filter combinations in FLUID-GPT are available in Figures S14-S15 in the Supporting Information. As observed in optimizing GPT-2, smaller stride yields improved MSE performance, albeit at the expense of longer training duration. This trade-off aligns with expectations, as a larger stride leads to larger steps between convolution operations, potentially affecting prediction accuracy.

However, a distinct relationship emerged for the kernel sizes in convolution and max pooling layers. Smaller convolution kernel sizes correlated with higher MSE values, while reduced max pooling kernel sizes were associated with lower MSEs. This phenomenon arises from the operational disparities between these layers. Smaller convolution kernels capture fewer features per operation, potentially resulting in information loss when critical components span a wider receptive field. Conversely, smaller pooling kernels keep more information by down-sampling less, which is favorable for tasks requiring precise localization and leads to diminished MSEs. By evaluating MSEs and training duration, we identified the optimal hyperparameter configuration: a convolution stride of 2, convolution kernel size of 3, pooling kernel size of 2, and an (8-16-32-32) convolution filter combination. Notably, this optimal configuration is also held for BiLSTM, reinforcing our adoption of the same CNN architecture for erosion prediction.

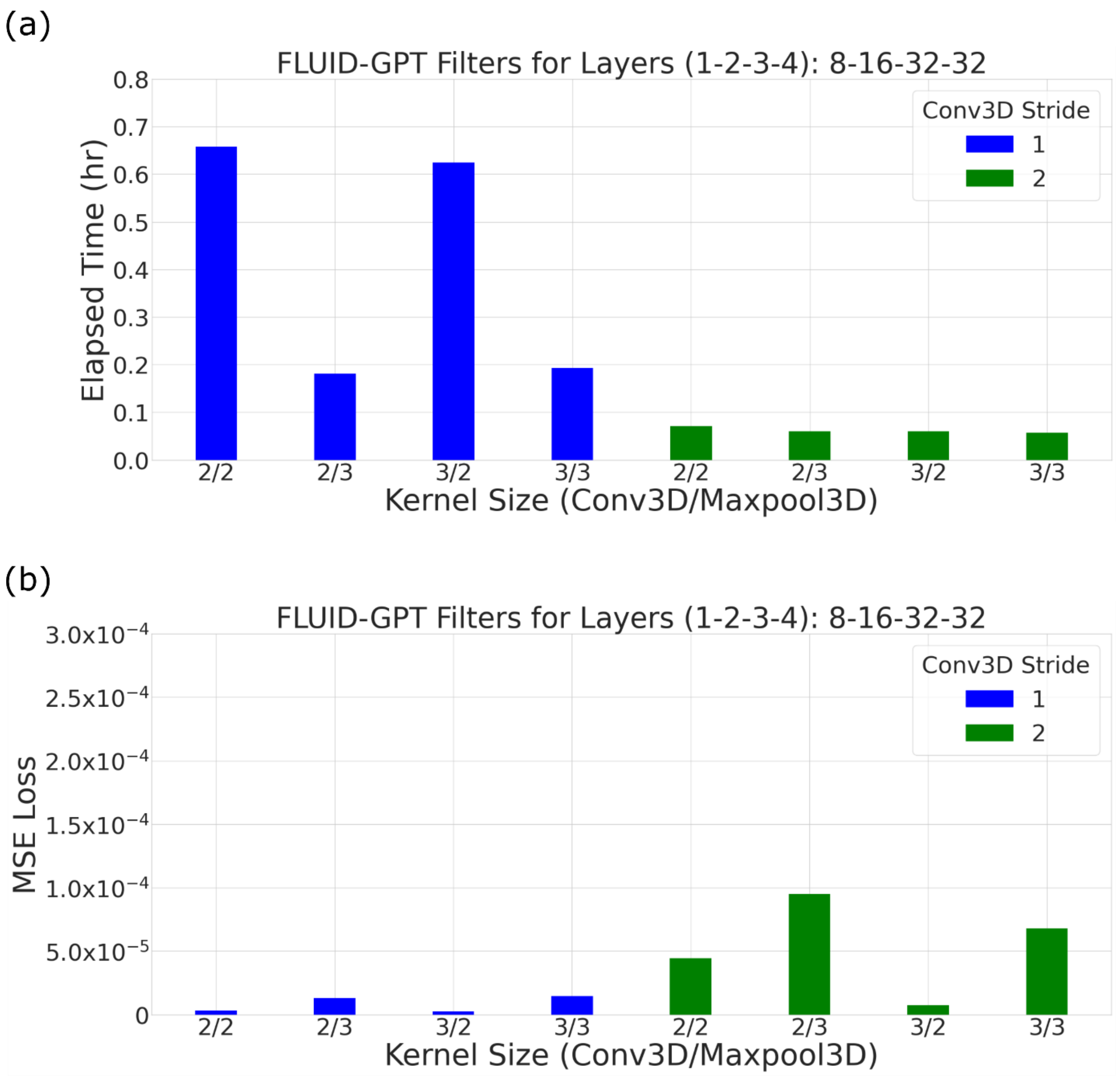


Figure 6. Optimizing CNN: Variations in stride and kernel sizes in the convolution layer and variations in kernel sizes in the max pooling layer with an (8-16-32-32) convolution filter combination. Panel (a) displays the training duration, while panel (b) shows the MSE loss validation for different hyperparameter values. The optimal hyperparameter configuration is a stride of 2, convolution kernel size of 3, pooling kernel size of 2, and an (8-16-32-32) convolution filter.

3.3 GPT-2 vs. BiLSTM Performance

After optimizing the GPT-2, BiLSTM, and their CNN models, we compared their prediction performance and training efficiency. Table S3 in the Supporting Information includes the results of the conventional LSTM model. Figures 7(a) and (b) show the training and validation loss of particle trajectory predictions from GPT-2 vs. BiLSTM and erosion predictions from FLUID-GPT vs. BiLSTM+CNN, respectively. Our early stopping criteria caused the GPT-2 training to stop after 18 epochs, while BiLSTM continued training up to 100 epochs. This preliminary stopping arises since GPT-2 divides the timesteps into segments with window and stride sizes. It subsequently processes them with multiple attention heads, allowing the model to review previous sequence information during each new segment. In contrast, the BiLSTM approach learns consecutively without recalling the previous timesteps. Reviewing past information by GPT-2 leads to more efficient learning but a longer training duration per epoch than BiLSTM, which resembles human learning. With early stopping, GPT-2 achieved faster convergence of the validating loss than BiLSTM, resulting in more accurate and efficient predictions.

We also compared the individual CNN models from GPT-2 and BiLSTM. However, despite having an identical CNN architecture, the BiLSTM+CNN model exhibits lower accuracy. Our results indicate that errors associated with the previous model's predicted trajectories may worsen the subsequent CNN performance. Evaluation using a test dataset and K-fold cross-validation yielded average MSEs, R^2 scores, and training durations (see Table 1). GPT-2 errors were nearly half the average MSE of BiLSTM, with better training efficiency. The average MSE of FLUID-GPT was 0.35 times lower than BiLSTM+CNN. The training duration was similar between FLUID-GPT and BiLSTM+CNN due to the shared CNN model.

Model	**MSE**	**R^2 Score**	**Training Duration**	
			Total (hr)	Epoch Time (min)
GPT-2	0.0053 ± 0.0004	0.9807 ± 0.0017	0.7362 ± 0.0474	2.4540
BiLSTM	0.0115 ± 0.0019	0.9427 ± 0.0102	2.4420 ± 0.1333	1.4652
FLUID-GPT	$7.2223 \times 10^{-6} \pm 5.0736 \times 10^{-7}$	0.9899 ± 0.0080	0.0651 ± 0.0016	0.0250
BiLSTM+CNN	$2.0650 \times 10^{-5} \pm 4.4573 \times 10^{-6}$	0.9707 ± 0.0061	0.0617 ± 0.0011	0.0246

Table 1. Comparison of prediction accuracy and training efficiency of GPT-2, BiLSTM, FLUID-GPT, and BiLSTM+CNN for particle trajectory and erosion predictions.

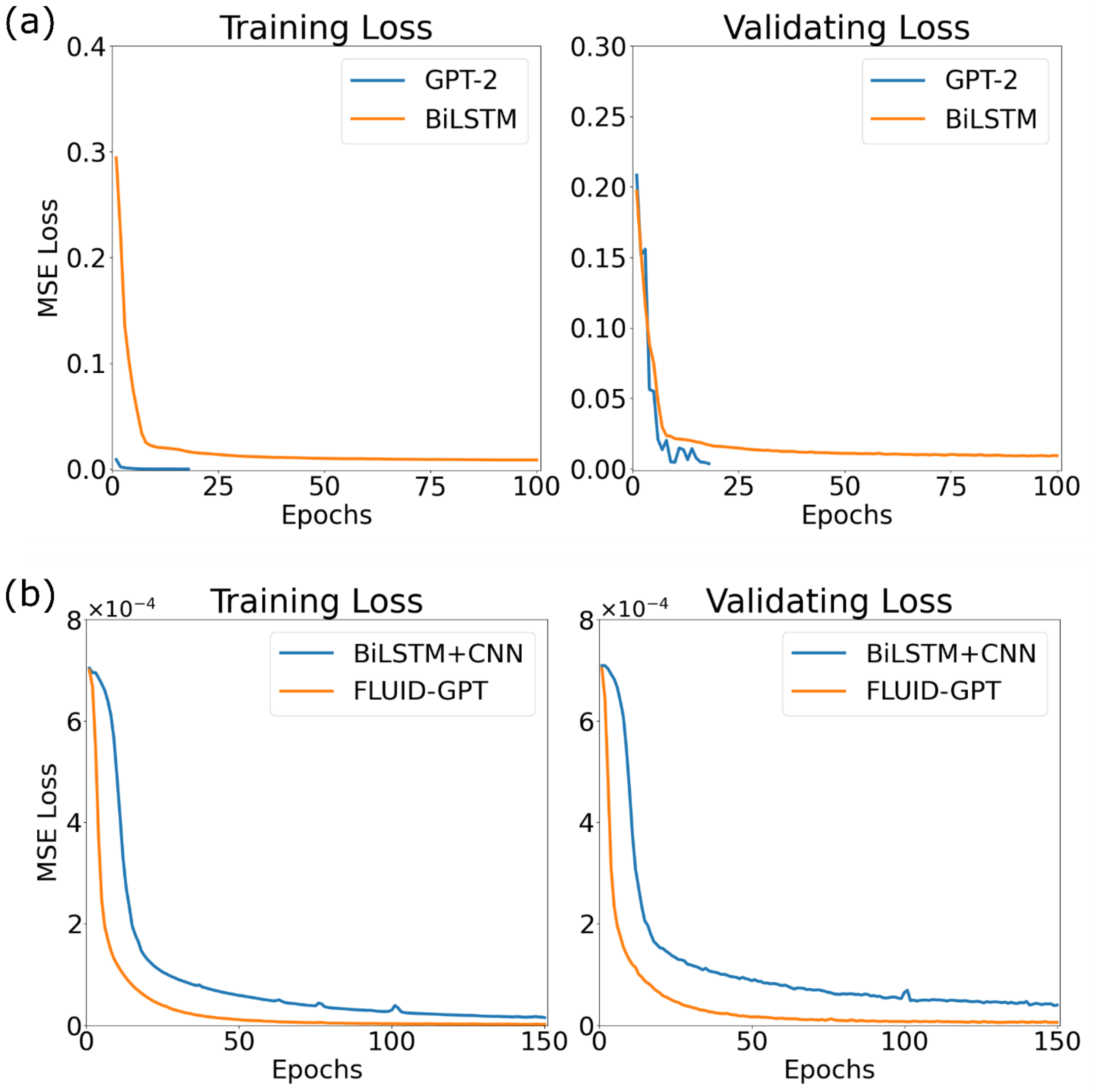


Figure 7. Training and validation loss comparison between (a) GPT-2 vs. BiLSTM, and (b) FLUID-GPT vs. BiLSTM+CNN, using optimized hyperparameters. The early stopping activates on GPT-2 training, converging much faster than BiLSTM. FLUID-GPT shows faster convergence, lower MSE, and a more stable learning profile than BiLSTM+CNN.

3.4 Performance in Predicting Trajectories and Erosion Rates

Figure 8 compares the predicted particle trajectories and surface erosion rates obtained from the brute-force CFD calculations and our optimized GPT-2 and BiLSTM models for two representative simulations (sample #50 and #70) from the test dataset. Figure 9 shows the results of our FLUID-GPT approach and BiLSTM+CNN, both of which use CNN on the output data for each respective time-series model. The conventional LSTM results are given in the Supporting Information (Figure S16) since they are similar to the BiLSTM results.

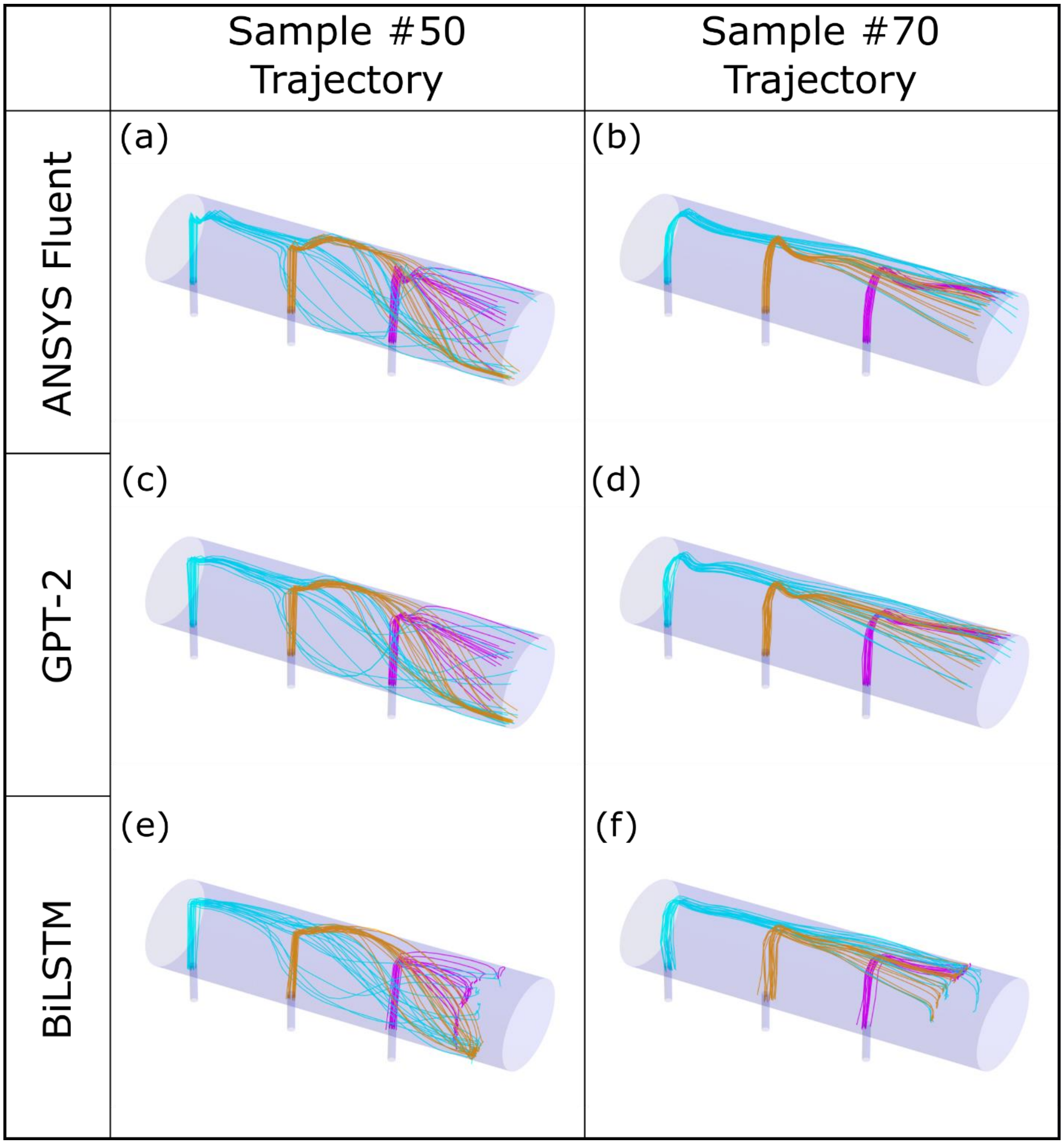


Figure 8. Comparison of particle trajectories predicted by GPT-2 and BiLSTM against ANSYS Fluent CFD simulations for representative simulations from the test dataset (samples #50 and #70).

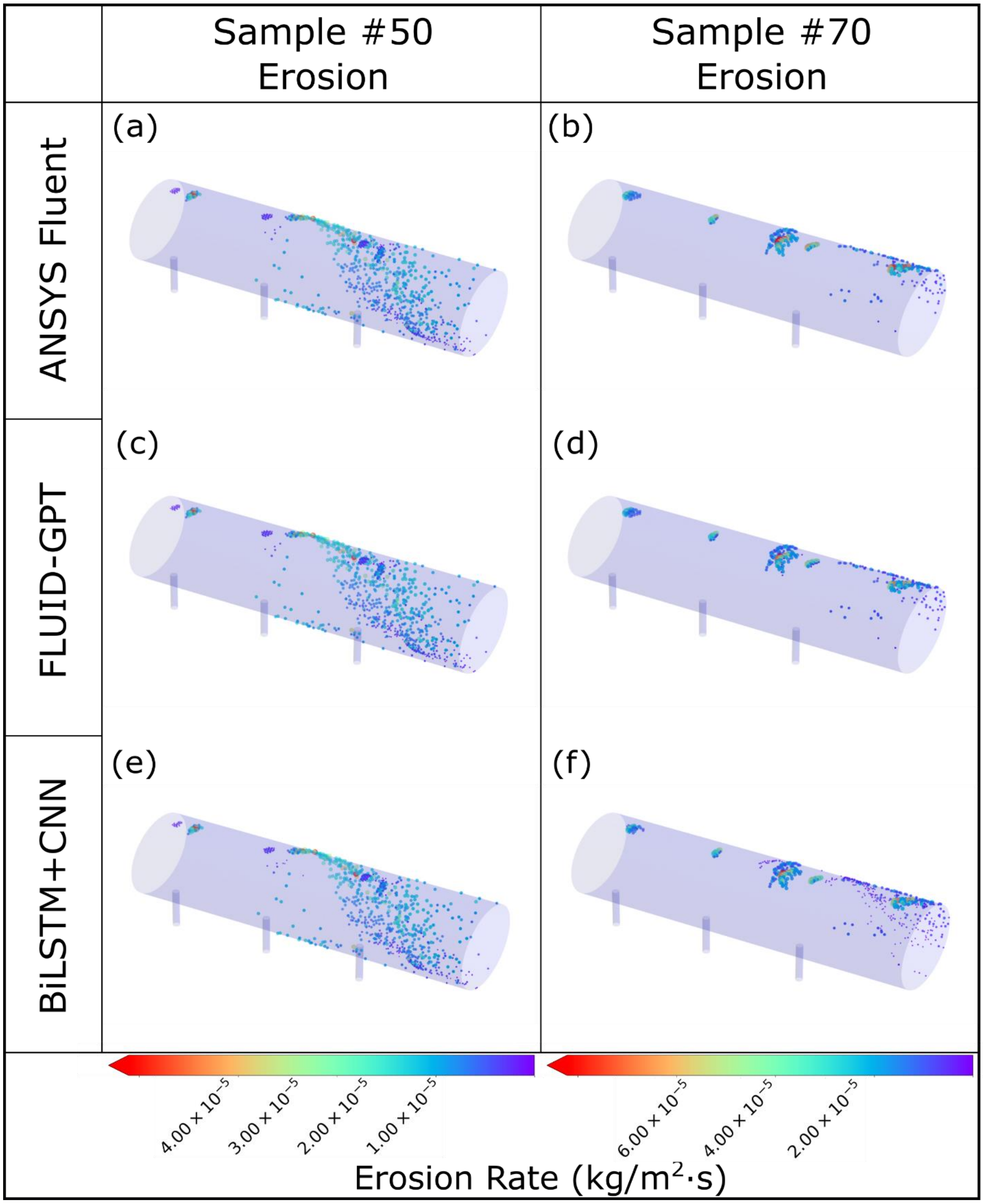


Figure 9. Comparison of surface erosion predicted by FLUID-GPT and BiLSTM+CNN against ANSYS Fluent CFD simulations for representative simulations from the test dataset (samples #50 and #70).

Figures 8(a)-(d) show that the GPT-2 model performs exceptionally well in predicting particle trajectories under turbulent flow conditions. In contrast, the trajectories predicted by BiLSTM exhibit noticeable deviations from the benchmark CFD data in Figures 8(e) and (f). These discrepancies can be attributed to the recurrent nature of the models. In the context of time-series data, GPT-2 is advantageous because it requires less contextual information and can capture dependencies among all timesteps. However, BiLSTM requires more details on the preceding and succeeding data points in the time series, which can be difficult to capture and may lead to lower accuracy. In addition, the forget gate in the BiLSTM algorithm can cause information loss from previous timesteps, further affecting prediction accuracy. Figures 9(a)-(d) highlight the good agreement between the FLUID-GPT and CFD benchmark calculations. Figures 9(e) and (f) present erosion predictions by the BiLSTM+CNN model, demonstrating satisfactory agreement in regions with significant erosion but displaying errors in areas with lower erosion than the CFD benchmarks.

In general, GPT-2 outperformed BiLSTM in both accuracy and efficiency, with a 54% decrease in average MSE (from 0.0115 to 0.0053) and a 70% reduction in training duration (from 2.45 hours to 0.73 hours). This advantage can be attributed to two algorithmic improvements in GPT-2: (1) information control, which segments the input data into windows and strides, enabling greater control over information mixing, and (2) autoregressive modeling, which considers all previous timesteps when predicting the next step. In contrast, BiLSTM processes input data sequentially and only predicts the subsequent step based on the previous step, resulting in lower accuracy and efficiency than GPT-2. Further deficiencies of BiLSTM can be attributed to the lack of information control and inability to consider all previous timesteps to predict the next step, which are algorithmic improvements implemented in GPT-2. For predicting erosion, the training duration of FLUID-GPT and BiLSTM+CNN is similar since the same CNN model is used.

However, the average MSE of FLUID-GPT improves by 65% compared to BiLSTM+CNN (from 2.0650×10^{-5} to 7.2223×10^{-6}). We attribute this improvement to the superior performance of GPT-2 in predicting particle trajectories. The BiLSTM model's initial and final segments of the predicted trajectories deviate from the CFD benchmarks, which negatively impacts erosion prediction accuracy. In contrast, the GPT-2 model performs exceptionally well in predicting particle trajectories, even under turbulent flow conditions, which allows the model to capture erosion features more accurately.

4. Conclusion

In conclusion, we have developed FLUID-GPT, a new hybrid GPT-2+CNN machine-learning architecture for accurately predicting particle trajectories and erosion on an industrial-scale steam header geometry. Our FLUID-GPT approach can predict surface erosion rates using only information from five initial conditions: particle size, main-inlet speed, main-inlet pressure, sub-inlet speed, and sub-inlet pressure. We optimized our GPT-2 model through systematic variation of learning rates and schedulers, implementation of early stopping criteria, and comprehensive analysis of hyperparameters such as the number of decoder layers, attention heads, and window/stride sizes. Furthermore, we refined the CNN component by fine-tuning kernel and stride sizes within convolutional layers, optimizing the kernel size for max pooling, and carefully selecting filter combinations across various convolutional layer variations. The training time for FLUID-GPT was approximately 47 minutes on a single GPU, with an impressive R^2 score of 0.98 and 0.99 for predicting particle trajectories and erosion, respectively.

Our study shows that the FLUID-GPT hybrid ML approach outperformed traditional time-series models such as LSTM and BiLSTM for predicting particle trajectories and erosion prediction. Specifically, GPT-2 showed a 54% decrease in MSE and was 70% faster than BiLSTM. For predicting erosion, FLUID-GPT showed a 65% improvement in MSE compared to BiLSTM+CNN. Our results demonstrate that the FLUID-GPT hybrid ML approach significantly improves upon our previous LSTM study for predicting trajectories and surface erosion. In particular, the GPT-2 algorithm yields impressive accuracy and has a fast-training duration compared to LSTM. Overall, our work demonstrates that FLUID-GPT is an accurate and efficient approach for predicting complex trajectories and their subsequent erosion patterns. As such, this approach could have promising widespread applications in other research areas requiring time-series analyses or predictions of complex spatial properties arising from time-dependent phenomena.

Supporting Information

Parameter settings for LSTM and BiLSTM, learning rate optimization with various schedulers for LSTM and BiLSTM, learning rate on GPT-2 with various schedulers, GPT-2 training duration and validating loss for 2, 3, and 4 transformer decoder layers as the number of attention heads is varied, FLUID-GPT and BiLSTM+CNN training duration and validation loss while varying the combination of filters, stride, kernel size of convolution layer, and the kernel size of max pooling layer, prediction performance and training efficiency of optimized GPT-2 and LSTM, and comparison of particle trajectories and erosion predicted by LSTM and LSTM+CNN against ANSYS Fluent CFD simulations for representative simulations from the test dataset.

Notes

The authors declare no competing financial interest. A Github repository containing all the CFD and ML models used in this work can be accessed at https://github.com/SDY159/CFD_ML2.

Acknowledgements

This work was supported by the U.S. Department of Energy, National Energy Technology Laboratory (NETL) under Award No. DE-FE0031746.

References

(1) Raghu, D.; McKee, B.; Wu, J. B. C.; Sheriff, C. High Temperature Erosion Resistant Materials for Petroleum Refinery Equipment. In *CORROSION 2001*; OnePetro, 2001.

(2) Pepi, M.; Squillacioti, R.; Pfledderer, L.; Phelps, A. Solid Particle Erosion Testing of Helicopter Rotor Blade Materials. *Journal of failure analysis and prevention* **2012**, *12* (1), 96–108.

(3) Swadźba, L.; Formanek, B.; Gabriel, H. M.; Liberski, P.; Podolski, P. Erosion-and Corrosion-Resistant Coatings for Aircraft Compressor Blades. *Surf Coat Technol* **1993**, *62* (1–3), 486–492.

(4) Vijiapurapu, S.; Cui, J.; Munukutla, S. CFD Application for Coal/Air Balancing in Power Plants. *Appl Math Model* **2006**, *30* (9), 854–866.

(5) Gandhi, M. B.; Vuthaluru, R.; Vuthaluru, H.; French, D.; Shah, K. CFD Based Prediction of Erosion Rate in Large Scale Wall-Fired Boiler. *Appl Therm Eng* **2012**, *42*, 90–100.

(6) Aramide, B. P.; Popoola, A. P. I.; Sadiku, E. R.; Aramide, F. O.; Jamiru, T.; Pityana, S. L. Wear-Resistant Metals and Composites. *Handbook of Nanomaterials and Nanocomposites for Energy and Environmental Applications* **2021**, 731–755.

(7) Calzolari, G.; Liu, W. Deep Learning to Replace, Improve, or Aid CFD Analysis in Built Environment Applications: A Review. *Build Environ* **2021**, *206*, 108315.

(8) Tao, J.; Sun, G. Application of Deep Learning Based Multi-Fidelity Surrogate Model to Robust Aerodynamic Design Optimization. *Aerosp Sci Technol* **2019**, *92*, 722–737.

(9) Morozova, N.; Trias, F. X.; Capdevila, R.; Schillaci, E.; Oliva, A. A CFD-Based Surrogate Model for Predicting Flow Parameters in a Ventilated Room Using Sensor Readings. *Energy Build* **2022**, *266*, 112146.

(10) Du, P.; Zhu, X.; Wang, J.-X. Deep Learning-Based Surrogate Model for Three-Dimensional Patient-Specific Computational Fluid Dynamics. *Physics of Fluids* **2022**, *34* (8), 081906.

(11) Pandya, D. A.; Dennis, B. H.; Russell, R. D. A Computational Fluid Dynamics Based Artificial Neural Network Model to Predict Solid Particle Erosion. *Wear* **2017**, *378*, 198–210.

(12) Zahedi, P.; Parvandeh, S.; Asgharpour, A.; McLaury, B. S.; Shirazi, S. A.; McKinney, B. A. Random Forest Regression Prediction of Solid Particle Erosion in Elbows. *Powder Technol* **2018**, *338*, 983–992.

(13) Tran, A.; Furlan, J. M.; Pagalthivarthi, K. V; Visintainer, R. J.; Wildey, T.; Wang, Y. WearGP: A Computationally Efficient Machine Learning Framework for Local Erosive Wear Predictions via Nodal Gaussian Processes. *Wear* **2019**, *422*, 9–26.

(14) Tran, A.; Wang, Y.; Furlan, J.; Pagalthivarthi, K. V; Garman, M.; Cutright, A.; Visintainer, R. WearGP: A UQ/ML Wear Prediction Framework for Slurry Pump Impellers and Casings. In *Fluids Engineering Division Summer Meeting*; American Society of Mechanical Engineers, 2020; Vol. 83723, p V002T04A008.

(15) Dai, W.; Mohammadi, S.; Cremaschi, S. A Hybrid Modeling Framework Using Dimensional Analysis for Erosion Predictions. *Comput Chem Eng* **2022**, *156*, 107577.

(16) Bakhshesh, M.; Bahrainian, S. S.; Hajidavalloo, E.; Parsi, M. Developing a Dimensionless Number for Solid Particle Erosion with Gas-Liquid-Solid Flow in Standard Elbows. *Wear* **2021**, *478*, 203769.

(17) Bahrainian, S. S.; Bakhshesh, M.; Hajidavalloo, E.; Parsi, M. A Novel Approach for Solid Particle Erosion Prediction Based on Gaussian Process Regression. *Wear* **2021**, *466*, 203549.

(18) Yang, S. D.; Ali, Z. A.; Kwon, H.; Wong, B. M. Predicting Complex Erosion Profiles in Steam Distribution Headers with Convolutional and Recurrent Neural Networks. *Ind Eng Chem Res* **2022**, *61* (24), 8520–8529.

(19) Radford, A.; Narasimhan, K.; Salimans, T.; Sutskever, I. Improving Language Understanding by Generative Pre-Training. *OpenAI blog* **2018**, 1 (8), 12.

(20) Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J. D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A. Language Models Are Few-Shot Learners. *Adv Neural Inf Process Syst* **2020**, *33*, 1877–1901.

(21) Geneva, N.; Zabaras, N. Transformers for Modeling Physical Systems. *Neural Networks* **2022**, *146*, 272–289.

(22) Shalova, A.; Oseledets, I. Deep Representation Learning for Dynamical Systems Modeling. *arXiv preprint arXiv:2002.05111* **2020-02-10** https://arxiv.org/abs/2002.05111 (Accessed 2023-07-05).

(23) Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; Polosukhin, I. Attention Is All You Need. *Adv Neural Inf Process Syst* **2017**, *30*.

(24) Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; Sutskever, I. Language Models Are Unsupervised Multitask Learners. *OpenAI blog* **2019**, *1* (8), 9.

(25) Xu, J.; Sun, X.; Zhang, Z.; Zhao, G.; Lin, J. Understanding and Improving Layer Normalization. *Adv Neural Inf Process Syst* **2019**, *32*.

(26) Li, C.; Zhang, M.; He, Y. Curriculum Learning: A Regularization Method for Efficient and Stable Billion-Scale Gpt Model Pre-Training. *arXiv preprint arXiv:2108.06084* **2021-08-13** https://arxiv.org/abs/2108.06084 (Accessed 2023-07-05).

(27) Meng, K.; Bau, D.; Andonian, A.; Belinkov, Y. Locating and Editing Factual Associations in Gpt. *Adv Neural Inf Process Syst* **2022**, *35*, 17359–17372.

(28) Maturana, D.; Scherer, S. Voxnet: A 3d Convolutional Neural Network for Real-Time Object Recognition. In *2015 IEEE/RSJ international conference on intelligent robots and systems (IROS)*; IEEE, 2015; pp 922–928.

(29) Taler, J.; Węglowski, B.; Taler, D.; Sobota, T.; Dzierwa, P.; Trojan, M.; Madejski, P.; Pilarczyk, M. Determination of Start-up Curves for a Boiler with Natural Circulation Based on the Analysis of Stress Distribution in Critical Pressure Components. *Energy* **2015**, *92*, 153–159.

(30) Modliński, N.; Madejski, P.; Janda, T.; Szczepanek, K.; Kordylewski, W. A Validation of Computational Fluid Dynamics Temperature Distribution Prediction in a Pulverized Coal Boiler with Acoustic Temperature Measurement. *Energy* **2015**, *92*, 77–86.

(31) Modliński, N.; Szczepanek, K.; Nabagło, D.; Madejski, P.; Modliński, Z. Mathematical Procedure for Predicting Tube Metal Temperature in the Second Stage Reheater of the Operating Flexibly Steam Boiler. *Appl Therm Eng* **2019**, *146*, 854–865.

(32) Horwitz, J. A. K.; Mani, A. Accurate Calculation of Stokes Drag for Point–Particle Tracking in Two-Way Coupled Flows. *J Comput Phys* **2016**, *318*, 85–109.

(33) Allen, D. M. The Relationship between Variable Selection and Data Augmentation and a Method for Prediction. *Technometrics* **1974**, *16* (1), 125–127.

(34) Wojtas, M.; Chen, K. Feature Importance Ranking for Deep Learning. *arXiv preprint arXiv:2010.08973* **2020-10-18** https://arxiv.org/abs/2010.08973 (Accessed 2023-07-05).

(35) Huang, Z.; Xu, W.; Yu, K. Bidirectional LSTM-CRF Models for Sequence Tagging. *arXiv preprint arXiv:1508.01991* **2015-08-09** https://arxiv.org/abs/1508.01991(Accessed 2023-07-05).

(36) Schuster, M.; Paliwal, K. K. Bidirectional Recurrent Neural Networks. *IEEE Transactions on Signal Processing* **1997**, *45* (11), 2673–2681.

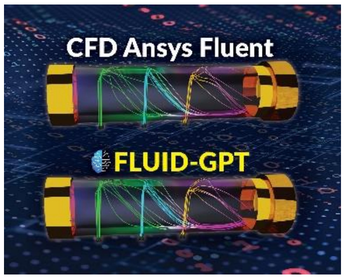
(37) Hendrycks, D.; Gimpel, K. Gaussian Error Linear Units (Gelus). *arXiv preprint arXiv:1606.08415* **2016-06-27** https://arxiv.org/abs/1606.08415 (Accessed 2023-07-05).

(38) Hendrycks, D.; Gimpel, K. Bridging Nonlinearities and Stochastic Regularizers with Gaussian Error Linear Units. *CoRR, abs/1606.08415* **2016**, *3*.

(39) Smith, L. N. Cyclical Learning Rates for Training Neural Networks. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*; IEEE, 2017; pp 464–472.

(40) Lee, H.; Lee, G.; Kim, J.; Cho, S.; Kim, D.; Yoo, D. Improving Multi-Fidelity Optimization with a Recurring Learning Rate for Hyperparameter Tuning. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*; 2023; pp 2309–2318.

(41) Loshchilov, I.; Hutter, F. Sgdr: Stochastic Gradient Descent with Warm Restarts. *arXiv preprint arXiv:1608.03983* **2016-08-13** https://arxiv.org/abs/1608.03983 (Accessed 2023-07-05).



For Table of Contents Only