Modelling and Optimizing Real-time Scientific Experimentation Using Transformer Generative Models and Reinforcement Learning

First Author[0000−1111−2222−3333]

School of XYZ, University of XYZ, XYZ, XYZ
xyz@xyz.ac

Abstract. Here we present an encoder-decoder architecture based on the Transformer model to simulate real-time scientific experimentation, predict its future behaviour and manipulate it on a step-by-step basis. The Transformer model was modified to use its encoder to process user-defined variables, and the decoder to model the actual experiment. As a proof of concept, this architecture was trained to map a set of mechanical inputs to the oscillations generated by a chemical reaction. To validate the trained model, we used it to recognize handwritten digits from the MNIST dataset, where it converted MNIST data into chemical oscillations, and these oscillations were decoded using a simple neural network to recover the original MNIST class. Finally, the model was paired with a Reinforcement Learning controller to show how the simulated chemistry can be manipulated in real-time towards user-defined behaviours. Our results demonstrate how Generative Learning can model real-time scientific experimentation to track how it changes through time as the user manipulates it, and how the trained models can be paired with optimisation algorithms to discover new phenomena beyond the physical limitations of lab experimentation. This work paves the way towards building surrogate systems where physical experimentation interacts with machine learning on a step-by-step basis, beyond the current focus on class predictability.

Keywords: Machine Learning in Science · Generative Modelling · Reinforcement Learning.

1 Introduction

Life and physical sciences have always been quick to adopt the latest advances in machine learning to accelerate scientific discovery. Today, Deep Learning is used to detect cancers from biomarkers [1], or achieve human-like results in problems such as cell segmentation [21]. Nevertheless, these exceptional results are based on mining previously created datasets to discover patterns or trends. Recent advances in AI have been demonstrated in real-time scenarios like self-driving cars [3] or playing video games [20]. However, these new techniques are struggling to be adopted in life or physical sciences because experimentation is slow, with lead time going from minutes to days.

To tackle this limitation, this work aims to adapt Generative Deep Learning algorithms to model scientific experiments and accelerate their discovery using *in-silico* simulations. We particularly focused on real-time experiments, aiming to model how they react to user inputs (Figure 1-A). In particular, we focused on modelling an oscillatory chemical reaction placed in a five-by-five array of weakly connected cells, and how the oscillations within each cell could be controlled using magnetic stirrers, see Figure 1-B. In this experiment the relation between magnetic stirrers and chemistry is not one to one, and there is no known model which can fully describe it. Therefore, compared with similar problems where the physical phenomena are modelled based on known systems, here we aimed to model it using only experimental data.

This work focused on the Transformer model (figure 1-C), among the several encoder-decoder architectures, since it is currently outputting very promising results in translation tasks [27]. The original model, which from now on will be referred as "vanilla Transformer", aimed at solving language translation problems with an encoder-decoder architecture that used "attention" layers. Originally, these layers were placed between encoder and decoder, and they were used by the decoder to decide which parts of the source sentence pay *attention* to [2]. The Transformer architecture extended this idea by building both encoder and encoder using self-attention layers to compute representations of its input and output [27]. This work shows how the Transformer model can be further modified to work with scientific data comprising of both user-defined variables, such as temperature or stirring patterns, and raw experimental data, such as the video of an on-going experiment. Finally, we focused on using the trained model with optimization or exploratory algorithms, such as Evolutionary Algorithms or Reinforcement Learning, to discover novel phenomena. The specific contributions of this work are:

- A first Transformer architecture to model and predict how a real-time scientific experiment changes over time based on user-defined inputs.
- A first Reinforcement Learning controller which consists of a single layer of neurons and uses the attention weights as returned by the Transformer's decoder to calculate its reward function.
- A demonstration of how the Transformer model can act as a kernel-like function to increase the experimental feature space.

2 Background and Related Work

Generative modelling aims to describe how a dataset is generated, in terms of a probabilistic model. By sampling from this model, it can generate new data [14]. This technique, paired with Deep Learning, has recently been used to teach machines how to write [28], paint [13] or compose music [8]. This work aims to use generative modelling to model scientific experiments for which no known theoretical model exists, but only experimental data. As an example of this kind of experiment, we focused on modelling the behaviour of a chemical oscillator,

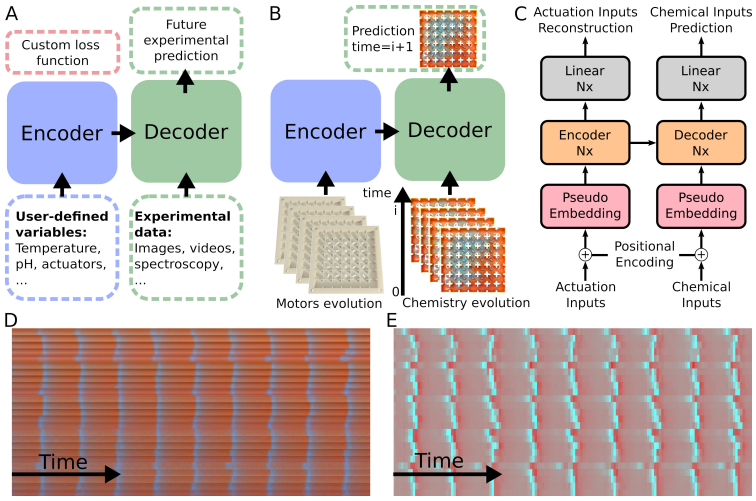

Fig. 1. A: This work proposes to use encoder-decoder architectures with experimental data. B: We focused on a chemical reaction that produces oscillations when stirred. The objective of this work was to map motor oscillations as defined by the user, to chemical oscillations as generated by the chemistry. C: Our encoder-decoder architecture used the Transformer model. The main variations are a reconstruction output layer in the encoder, and a linear output used for regression in the decoder. D: Chemical oscillations as generated by a physical platform. E: Chemical oscillations as generated by our model.

which is a type of chemical reaction which continuously oscillates between two or more states [11].the In particular, we focused on modelling how the oscillations can be manipulated via user input [10]. This type of experiment has been tried to be modelled using theoretical models, such as the Oregonator model [17]. However, this model is a simplistic approximation and it fails to capture the dynamics of real world complex experiments, like the chemical oscillator described in [23].

To achieve our aims, we focused on encoder-decoder architectures, which are commonly used in natural language processing tasks [28]. In machine translation, for example, a sentence in language "A" is processed through the encoder into a fixed-length vector from which the decoder generates a translation in language "B". In a similar way this work aims to process user-defined data through the encoder to generate a latent space, and then use this latent space in the decoder to generate experimental data. Among the different encoder-decoder architectures, we focused on the Transformer model [27]. Therefore, the main technical objective of this work was to adapt the Transformer architecture to work with experimental data.

The objective of this work is related to recent work in the field of physics where machine learning has been used to model physical phenomena from known models using graph networks [24], encoder-decoder recurrent neural networks (RNN) [29, 5] or Generative Adversarial Networks [30]. As opposed to physics,

chemistry rarely has known theoretical models, and chemists usually work using their own intuition and empirical evidence. Based on this, chemists have developed the field of "chemoinformatics" where machine learning has been used extensively [19, 26], and recently generative architectures similar to the ones developed in this work have also been used [6, 12, 4, 25]. The main difference in this work is that we did not aim to predict synthesizability, but we focused instead on tracking real-time experimental changes based on user-input. Similarly to chemistry, machine learning and generative modelling have been used in biology [7]. AI has also been used to guide experimentation, either carried out by researchers [9] or an automated platform [22, 15]. Nevertheless, in these publications machine learning was used to propose experiments that were tested in the real world. On the other hand, this work aimed to simulate *in-silico* the experiment. This way, this research is related to the idea of "dream environments" where an agent explores a simulated environment to then transfer its knowledge into the real one [16].

3 Results

3.1 Data collection and processing

In related work [23] the authors built a platform to manipulate a chemical oscillator in a five-by-five array of weakly connected cells. Each cell contained a magnetic stirrer that was rotated using a motor. These rotations stirred the chemistry, which eventually generated oscillations that were related to the configuration of motors used: which motors were enabled, their directions, and their speeds. While they succeeded in using the platform to perform pattern recognition, the results were limited due to the low data throughput of real-world experimentation. It is the objective of this research to address this limitation by using Deep Generative Learning to model and study the chemistry.

Using the described physical platform, a series experiments were performed where different combinations of motors were activated at different times. The objective was to study both how a given motor configuration gave rise to different chemical oscillations, and how the oscillatory nature of the chemical reaction would impact newer motor configurations and therefore new chemical oscillations, as oscillations can continue to be observed even after stirring is disabled. For each experiment we saved a video of the experiment, and the motor configuration used at every time-step. Finally, the data was split into sequences. See the Methods section (Section 5.1) for a full description of the data preprocessing.

3.2 Using the Transformer architecture for scientific experimental regression

Compared to the "vanilla Transformer" implementation, the four main differences in this work are (Figure 1-C): (1) The decoder softmax layer was replaced with a dense layer with same size as the decoder's input feature space. This way,

it could solve regression problems as the one we were targeting. (2) The embedding layers were replaced with dense layers which transformed the encoder and decoder inputs into their respective latent spaces. (3) We added a new output to the encoder, which aimed to reconstruct its input. This reconstruction was added because very often lab experiments have some degree of inertia, meaning that there is a lag between user inputs and how the experiment behaves. We found that without forcing the encoder reconstruction as part of the learning process, the decoder would ignore the encoder (ie. user defined variables), and it was predicting the next state of the experiment using only the previous states of the experiment (the decoder input). (4) Both the encoder and the whole Transformer were trained in cycles, where initially the encoder was trained for a few iterations, and then the whole Transformer, including the encoder, was trained for more iterations. See the Methods (Section 5.2) for implementation details.

Using the described architecture, our Transformer model was trained to learn associate sequences of motor actuations to sequences of experimental data, and then use this association to predict subsequent behaviour. The Transformer's attention mechanism was critical for the success of this task, because the experimental data represented oscillations that might appear for one or two seconds, with a periodicity of 40 to 60 seconds, meaning that for more than 95% of the time, the signal to train for was zero. Attention layers can remember that there was an oscillation back in time. Figure 2 shows some of the Transformer predictions compared with the original data as generated by the physical platform. Initially, the Transformer replicated the original data with a very high degree of accuracy, but as the experiment evolved, prediction and real experimental data begin to diverge. Nevertheless, the Transformer still produced outputs that resembled an oscillatory chemical reaction. It is important to say that, even in a real experiment using the automated platform, the chemistry never behaved the same way, and the same experiment repeated twice could output different data.

These results show that the described model can, to some degree, generate data similar to physical platform The advantage of this model resides on its data throughput. While the physical platform could generate at most one experiment per hour, the simulated platform using the Transformer model could generate many more experiments per second. This enabled us to use the simulated platform paired with different optimization or exploratory algorithms to discover new phenomena, in a way that was not possible before.

3.3 Performing pattern recognition using the learned model

In related work, the described platform was used to perform pattern recognition [23]. To do so, a pattern was mapped into the five-by-five array of motors. This would generate chemical oscillations related to the input pattern, and then these oscillations were decoded using a neural network. Unfortunately, each experiment took one hour, and thus the training dataset was small.

Using the trained Transformer model a similar process was followed, but instead of using a real platform to generate oscillations, we used the trained model, which had a much higher data throughput. To test this, we used the

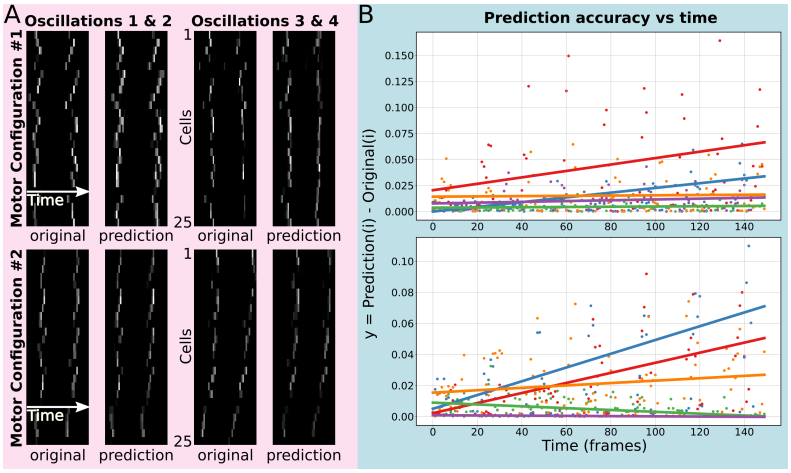

Fig. 2. A: Comparing the first four oscillations – from two different motor configurations – as generated using the Transformer model with data from the test set. B: Comparing the prediction generated using the Transformer model with data from the test set over a full experiment. Each plot shows five different experiments. The oscillations at time i in the original (test) dataset were compared with the oscillations in the prediction at time $i \pm w$ to account for phase differences. The accuracy represented in the y axis is per cell. See Supplementary Movies 1-2 for videos of the experiments.

MNIST dataset, from where we created two smaller datasets, one containing the numbers from 0 to 2, and the other one from 0 to 4. Each MNIST data point was pre-processed and resized into a five-by-five image, and then each of the 25-pixel values was associated to the speed of one of the 25 motors in the Transformer model (see Section 5.3 for full implementation details). The model then created a sequence with 300 time-steps. These sequences were then divided into windows of size W, and these windows were connected to a single dense layer of size N, followed by an output layer, which was used to decode the original oscillations, see Figure 3.

Following this approach, the single layer of neurons that received windows of data generated from the Transformer, had an accuracy over 80% when testing for three classes, and almost 70% when training for five classes. Although these test accuracies are very low when compared to other MNIST results, which can achieve accuracies near a 99% when testing for all the numbers [18], it still proves that: (1) the platform has the potential to encode complex datasets given enough time and resources, and (2) the Transformer model produced data that generated similar oscillations for similar inputs in a way that they could be classified by a very simple neural network.

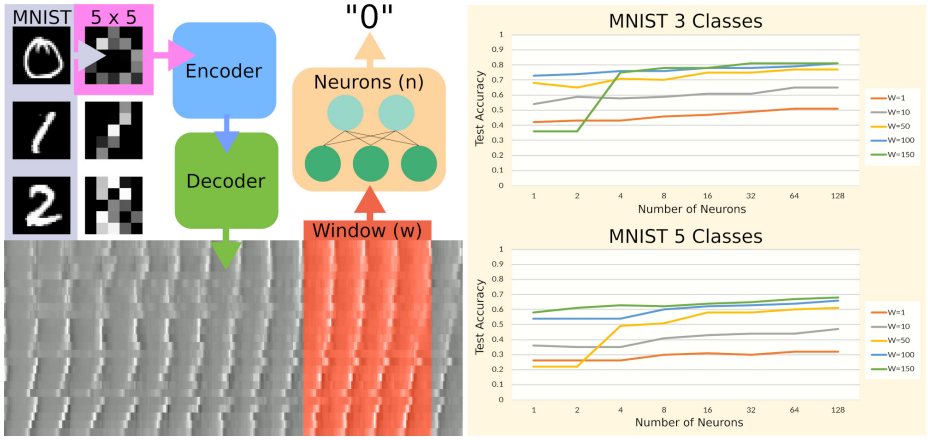

Fig. 3. Initially, MNIST data is resized into five-by-five images. The "motors" in the decoder are set to speeds matching the resized image, and the decoder is used to generate oscillations. Windows of oscillations are fetched, and a single layer neural network is used to decode them into the original MNIST number. This process was repeated for three (0,1,2) and five (0,1,2,4,5) MNIST classes.

3.4 Pairing the Transformer model with a Genetic Algorithm

Once shown that the trained Transformer model could generate oscillations based on user-defined motors inputs, the next step was to pair the model with a Genetic Algorithm (GA) to find configurations of motor that manipulated the chemistry to behave in a user-defined way. In particular, the GA was used to find motor configurations where the chemistry "behaved" like an XOR gate. To accomplish, the first and fifth rows of the 5 by 5 array were defined as the XOR inputs, and centre cell as the XOR output (Figure 4). The inputs could be set to either 0, meaning the motors did not rotate, or 1, meaning all the motors in that row rotated at full speed. The output was considered 1 if the centre cell oscillated more, on average, than the other 24 cells, and it was considered 0 if it oscillated less. Here by "oscillation" we meant the cell value as outputted by the Transformer – which was training using blue channel data. Therefore, higher values mean the blue colour was stronger, which is associated with an oscillation, while no oscillations is associated with red colour.

Following this, the GA aimed to find motor configurations for the motors for rows two, three and four, whereby enabling or disabling the first and fifth row, the output (centre cell) behaved like an XOR for its four cases. That is, the centre cell would oscillate less than the other cells if the first and fifth row were both either enabled (11) or disabled (00), and it would oscillate more if one of them was enabled and the other disabled (01 and 10), see Figure 4 and Supplementary Movie 3. It is important to note that the GA found single motor configurations that worked for the four XOR cases: the speed of the motors between the second and fourth row was the same regardless of the inputs.

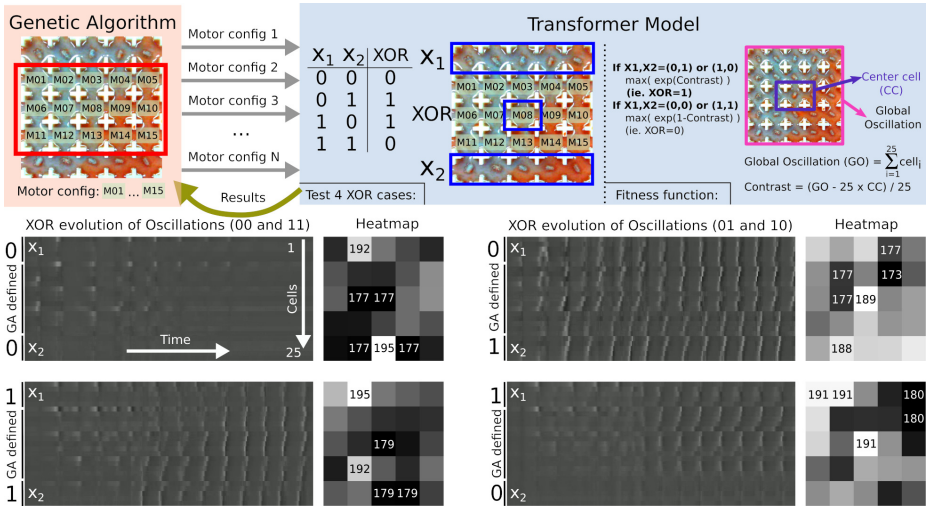

Fig. 4. A GA was used to generate motor oscillations that would activate the chemistry to oscillate like an XOR gate. To do so, the first and fifth row were used as inputs, while the centre cell was used as output. These motor oscillations would then be tested using the trained Transformer model. The bottom side of the figure shows a particular motor configuration and how it emulates a XOR. The results can be seen in the heatmaps, where the centre cell behaves as expected based on the inputs.

3.5 Using Reinforcement Learning to optimise chemical phenomena

One of the main features of the original platform is that different motor speeds can be applied at any time. In this kind of problems, where the "action" can change as the problem evolves, Reinforcement Learning (RL) can offer better results than a GA as used before. Following the RL implementation from [16], our controller was a single layer of neurons that was directly connected to the output of the decoder (Figure 5-A). This very simple controller took advantage of the fact that the Transformer model had already been trained to predict how the data would behave, in a similar way to how in [16] a single-layer neural network is used as Controller, receiving the output from a RNN. Our controller was trained to output the action (i.e. the motor configuration) needed to "push" the chemistry towards a user-defined behaviour using backpropagation: the whole model was used on the forward pass, but only the controller weights were modified.

To evaluate the quality of a motor configuration outputted by the controller, the new motor configuration was added to the "actuation inputs" that were used on the forward pass to output the given motor configuration, and these new actuation inputs plus the old chemical inputs were feed forward the transformer model described in Figure 1-C. This would output a new chemical prediction which would reflect how the new motor configuration impacted the behaviour of the chemistry (Figure 5-B). In this work, the task to solve by the "agent" was be to find motor configurations that maximized or minimized the value of

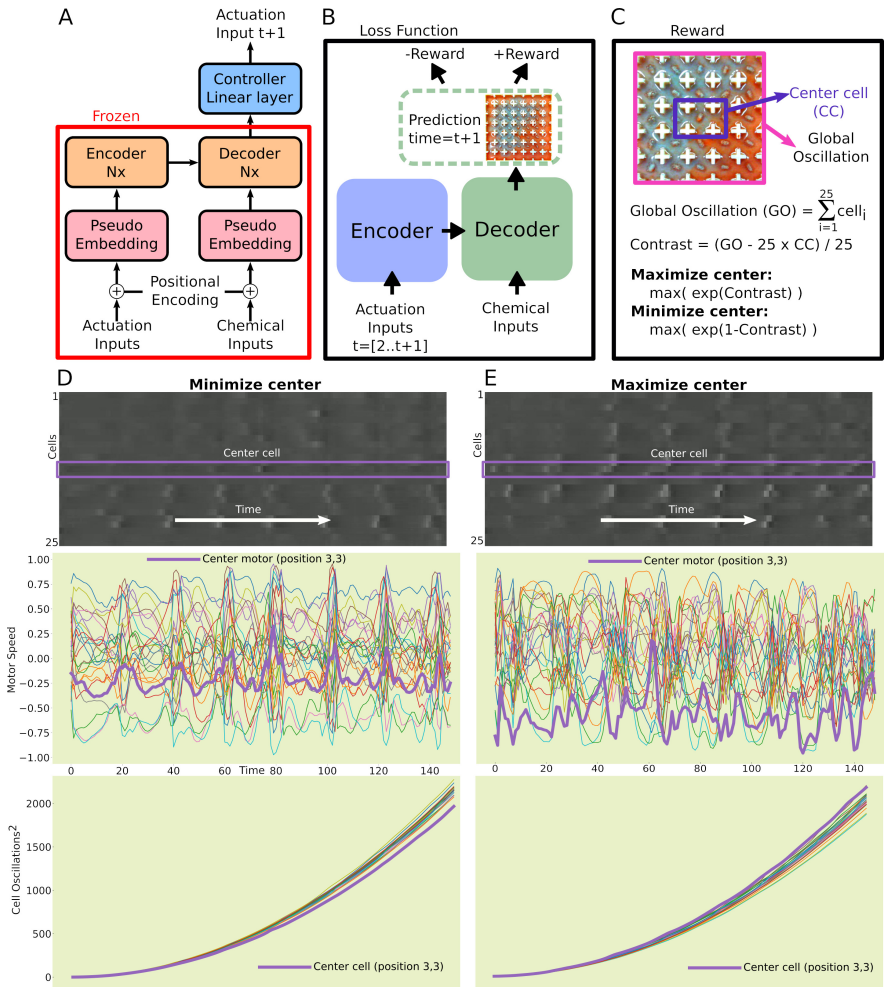

Fig. 5. A: To implement Reinforcement Learning, we will connect the output of the encoder to a single layer of neurons which will act as controller. B: The loss function will test the motor configuration as provided by the controller using the trained Transformer model. C: The motor configuration will be given a positive reward if it increases the oscillations of the centre cell. There is also an opposed case where the controller is trained to decrease the oscillations. D: Example of using the controller to maximize the centre cell. There it can be seen how the controller tweaks the motors before and after oscillations, and how the centre has, in average, a lower value than the others. E: Similar example but using the controller to maximize the centre cell. In both D and E, a motor speed of 0 means the motor is not moving, a motor speed of 1 means full speed clockwise, while a speed of 0 means full speed counter-clockwise. The two bottom plots named "Cell oscillations^2" describe the blue channel of each cell as the experiment progressed. It was squared for better visualization.

the centre cell against the other 24 cells in the array (Figure 5-C). Because the value of a cell is related to its blue channel, and higher values of blue channel represent oscillations, then the aim was to keep the centre cell oscillations above (or below) average.

Figure 5-D shows an example where the centre cell was minimized. It can be seen in the greyscale plot of cells oscillation vs time how it managed to suppress most oscillations. Below this plot the evolution of the motor speeds through time is shown, with a focus on the motor in position 3,3, which is the centre cell itself. Here we see how the controller was continually tweaking the value of the motors. Interestingly, the motor speeds oscillated in a similar way to the chemical reaction, with sudden changes when the cell actually oscillated. The bottom plot shows how the value of the centre cell was lower, on average, than all the others. Figure 5-E is similar to the D panel, but focuses instead on maximizing the centre cell. Both D and E started from the same state: Given the same chemical state, the controller managed to either maximize or minimize the objective function. See Supplementary Movie 4 for a video showing the results.

3.6 Using the Transformer model to increase the experimental feature space

One of the main limitations of physical experimentation is the size of the feature space used to represent both input and output variables. Even when using automated platforms connected to a computer that can handle a large feature space, the number of variables that can really be explored is limited by time and material constraints. An example of this limitation can be seen on the platform used to collect the data used in this research: the complexity of chemical phenomena that can occur in such a small feature space (5 by 5 oscillation outputs) was limited. In this section it will be explored how generative learning can be used to increase the feature space.

To do so, our Transformer model, which had 5-by-5 sequences of data as the I/O, was used as pseudo-kernel function to generate outputs of size N-by-N based on inputs of size N-by-N (Figure 6-A). To achieve this, the Transformer model was convolved along the N-by-N data inputs, in a similar way to Convolutional Neural Networks (CNN). While CNNs convolve along multiple channels, our model convolved along multiple time slices, and where CNNs generate a convolution output by performing arithmetic operations between the kernel and the input data, our model generated an output by feed-forwarding the input data through the Transformer model (Figure 6-B). To aid simplicity, given a 5-by-5 output as returned by the model, we only retained the centre cell, of size 1-by-1, and discarded the other 24 cells. If the target feature space is of size N by N, then a total of N x N operations will be needed (Figure 6-C).

Using this approach, we were able to generate an output space of, for example, size 50 by 50, which used a feature space of the same size as input (Figure 6-D). An interesting observation was how the simulation generated several "oscillatory fronts", which are very common when working with this chemistry, but that we never encountered in the 5-by-5 platform. Figure 6-E shows several of these

single time-steps flattened into a 1D, and concatenated. To see the original 50-by-50 video check Supplementary Movie 5. The resolution and complexity of this time-map is much higher than the ones we have presented before.

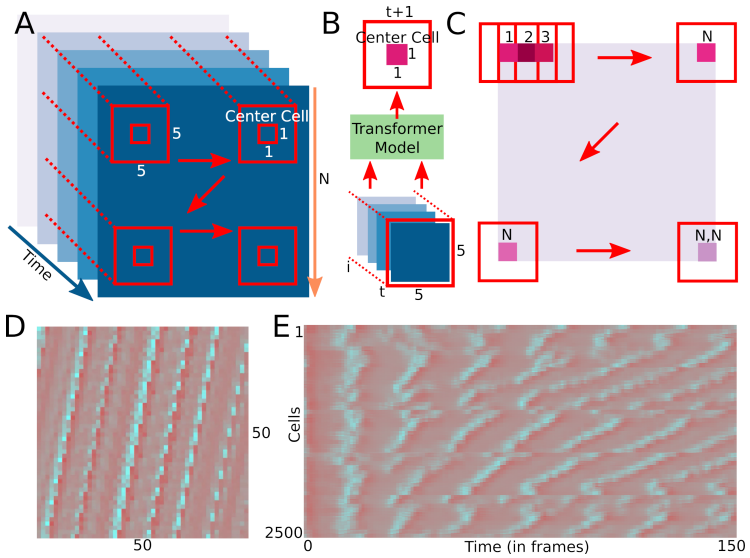

Fig. 6. A: Different slices of time oscillations are taking, and inputted into B the trained transformer model, which acts similarly to a kernel. C: From the output of the transformer, only the centre cell is kept, and the results are stitched together to generate an output of size N by N. D: Following this, oscillations of size 25 by 25 were created. E: The 25 by 25 oscillations plotted against time.

3.7 Limitations and discussion

The main limitation of the work is that while the quality of our modified Transformer model was validated against the test dataset, the quality of the results obtained using the GA and RL were not validated in a real platform, but the validation of their results was tested *in-silico*. The dataset we chose to test our approach (a chemical oscillator) was suitable due to being a small and easy to train, while offering distinctive properties, and being a regression problem. Nevertheless, this type of chemistry has a high degree of stochasticity, to the point that in our dataset two experiments with the same inputs produced different results. Considering the size of the dataset, and the stochasticity of the chemistry, it is unlikely that the results presented here could be transplanted to a real platform, but we think that this work shows the potential to do so. As such, it is our objective in future research to choose a more deterministic system, which still doesn't have a theoretical models, and fully validate the results.

From an implementation perspective, the main limitation of the framework presented here is the use of the Transformer model to calculate the loss function of the RL controller. This makes the loss function expensive to calculate. The second main technical limitation of our approach is related to the use of the Transformer model as a kernel to increase the feature space. In our implementation, the number of predictions needed is equal to the size of the target feature space. In the example described on Figure 6, for example, every snapshot needed 2500 predictions, and this means that to generate 300 timesteps our computer needed more than two hours.

4 Conclusion

In this work we presented a modified Transformer architecture to handle real-time scientific experiments. To do so, we explored how the Transformer's encoder could be used to interpret user-defined variables, while the decoder was used to simulate a scientific experiments. It is, to our knowledge, the first purely generative model applied to scientific experimentation, with a focus on generating real-time sequential data. We then showed how this model can interact with optimization or exploratory algorithms, such as Genetic Algorithms or Reinforcement Learning. Taking advantage of the high data throughput of our model, we could pair it with the described algorithms to discover new phenomena in a way that was not be possible before due to the limitations of physical experimentation

We are excited about the future of generative modelling applied to scientific experimentation beyond the current focus on class predictability. In particular when combined with automated platforms which can manipulate experiments in real-time. It is our future objective to apply the results described in this work on fully surrogate systems, where a robot would perform a physical experiment while the generative models perform high throughput *in-silico* experimentation that is continuously validated against the real experiments. We also plan to apply the described framework to many different experiments that would benefit from real-time monitoring, beyond the discussed oscillatory chemical reaction.

Source Code Included with the supplementary materials.

Acknowledgements We are grateful to Prof. Leroy Cronin for his fruitful comments and inspiration, and for providing the dataset used in this work.

5 Methods and materials

5.1 Data preparation

Using the physical platform described in [23], 55 different experiments were performed. This platform contained an array of 5 by 5 weakly connected cells, where the fluid and chemistry could move through them. Therefore, oscillations appearing in a cell could move to neighbouring cells. Each cell contained a magnetic

stirrer, and below each cell there was a DC motor with a magnet attached to its shaft. When any of the motors were actuated, the stirrer above them would also rotate, and thus stir the chemistry, which would eventually generate oscillations. Each experiment lasted 30 minutes where different motors were enabled at different speeds during sequences of one or two minutes. Thus, each experiment tested 15 or 30 motor combinations. To perform these experiments, the motor configurations to be tested were chosen either randomly or following user-defined shapes. In the user-defined experiments, we defined simple shapes, like a column of motors enabled while all the other ones were disabled.

Once all the experiments were performed, the data was stored as tuples with as many entries as frames in the video, where one element was the input motor patterns, and the other element was the associated chemical state, as recorded from a camera placed in top of the platform. The videos, which represented the evolution of the chemistry through the experiment, were processed in two different ways: (1) They were binarised into arrays with 25 cells of 0s and 1s. A 0 was assigned when no oscillation happened in that cell, while a 1 was assigned when an oscillation happened. The cell binarisation based on the colour of the chemistry was performed using a Super Vector Machine, as done in [23]. (2) For every frame, which was recorded using the RGB colour-scheme, the blue channel was isolated. Then, a moving average of the blue value was calculated for each of the 25 cells, a it was subtracted from the average value to centre the signal around zero and dampen the color changes of the chemical reaction as the experiment progresses, while maintaining the oscillations. Finally, it was normalized between 0 and 1.

The data related to the motors configurations was normalized between -1 (max speed counter-clockwise) and 1 (max speed clockwise), being 0 no speed (motor disabled). Before processing, every video was sped up by a factor of five, outputting 7200 frames. Then, we generated sequences of 150 elements, where each sequenced sampled the 7200 frames every 8 frames, with a stride of one. The last step converted the overall data into batches of size 64. Therefore, the final data had a shape like (64,150,25) for both the motor configurations and the chemistry evolution.

5.2 Transformer architecture and training

Architecture: Dense layers were used instead of embedding layers. The input data was already in the format (seq_length, input_features). Thus, these dense layer transformed it into (seq_length, latent_space_length). We obtained the best results using dense layers of size 128. The last "softmax" layer in the "vanilla Transformer" was replaced with a "relu" dense layer – or a "sigmoid" one in case of using the binarized dataset. We obtained the best results with a dense layer of size 1024 between the output of the last decoder layer and the final output dense layer – which was of size 25 since that's the size of our experimental feature space. The encoder output had two similar extra dense layers, one with size 1024, and the output one with an "elu" activation function. Finally, in our implementation

we obtained the best results with 4 stacked encoders and decoders (the "vanilla Transformer" uses 6).

Training: The model was trained with Adam optimizer, same parameters are "vanilla Transformer". The loss function used was a slightly modified "mean squared error" (mse). Ror every time-step in the sequence we calculated its mse, and also the maximum element from "y_true". Then we divided the time-step related mse with the time-step related max. Finally, the average per time-step was returned for the whole sequence We applied this small scaling factor to the mse because our dataset represents a chemical oscillator which has a value of 0 most the time, and we wanted the learning to particularly focus when an oscillation appears. In our implementation, the encoder was trained for 30 iterations, followed by 100 iterations where the whole Transformer was trained. This was iterated 10 times.

5.3 MNIST data to Transformer model

From the MNIST dataset, we created two smaller datasets, one containing the numbers from 0 to 2, and the other one from 0 to 4. The original MNIST dataset contained images of size 28 by 28. To input them into our model, first we removed a pad of three pixels around the boundary, and then we resized them to 5 by 5 using linear interpolation. Then, the pixel values were normalized in the range from 0.1 to 1. Once the MNIST dataset was pre-processed this way, a total of 10240 entries were randomly chosen, their values were set as the motor speed in the Transformer model, and a sequence with 300 oscillations was generated. These sequences were then divided into windows of size W, and these windows were connected to a single dense layer of size N, followed by an output layer of size 3 (or 5), which was used to decode the original oscillations.

5.4 GA and RL controller implementation

GA implementation: The platform consists of five rows with five motors. Rows one and five were left for I/O, therefore the GA will optimize rows two, three and four, for a total of 15 motors. In our GA implementation, each motor was considered a gene, therefore the genome was an array with 15 elements. Our GA followed the standard approach with the following operations: ranking, selection using the roulette algorithm, crossover (two point) and mutation. Our GA used a population size of 512, with 50 elite individuals, 100 generations and a mutation rate of 0.05.

RL controller: It was a single layer of 1024 neurons, which was connected to an output layer with 25 neurons - equal to the number of motors, using a sigmoid activation function. Given a sequence of motors and a sequence describing how the experiment changes, the first step was to feed-forward these two sequences through the Transformer model with the new controller output, to predict a new sequence of motors which would manipulate the chemistry towards a user defined target. The loss function of the controller received three variables: (1) the sequence of motors just described, (2) the sequence describing the experiment as

just described and (3) the motor configuration prediction just outputted by the controller. The loss function would then concatenate the new motor prediction (3) to the old sequence of motors (1), removing the oldest element to keep a sequence of constant length – which in our case was 150. Then we would input this new motors sequence (1+3) with the sequence describing the experiment (2) through the Transformer described in section 5.2 to predict a new chemical state. Based on this chemical state, we would consider if the new motor configuration (3) had a positive or a negative effect against the user defined target function, and then reward it accordingly. Using this loss function the controller layer was trained using backpropagation.

References

1. Ardila, D., Kiraly, A.P., Bharadwaj, S., Choi, B., Reicher, J.J., Peng, L., Tse, D., Etemadi, M., Ye, W., Corrado, G., Naidich, D.P., Shetty, S.: End-to-end lung cancer screening with three-dimensional deep learning on low-dose chest computed tomography. Nat. Med. **25**(6), 954–961 (2019)
2. Bahdanau, D., Cho, K.H., Bengio, Y.: Neural machine translation by jointly learning to align and translate. Int. Conf. Learn. Represent. (2015)
3. Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L.D., Monfort, M., Muller, U., Zhang, J., Zhang, X., Zhao, J., Zieba, K.: End to End Learning for Self-Driving Cars. arXiv **1604.07316** (2016)
4. Bort, W., Baskin, I.I., Gimadiev, T., Mukanov, A., Nugmanov, R., Sidorov, P., Marcou, G., Horvath, D., Klimchuk, O., Madzhidov, T., Varnek, A.: Discovery of novel chemical reactions by deep generative recurrent neural network. Sci. Rep. **11**, 3178 (2021)
5. Colen, J., Han, M., Zhang, R., Redford, S.A., Lemma, L.M., Morgan, L., Ruijgrok, P.V., Adkins, R., Bryant, Z., Dogic, Z., Gardel, M.L., de Pablo, J.J., Vitelli, V.: Machine learning active-nematic hydrodynamics. Proc. Natl. Acad. Sci. U. S. A. **118**(10) (2021)
6. Coley, C.W., Jin, W., Rogers, L., Jamison, T.F., Jaakkola, T.S., Green, W.H., Barzilay, R., Jensen, K.F.: A graph-convolutional neural network model for the prediction of chemical reactivity. Chem. Sci. **10**(2), 370–377 (2019)
7. Das, P., Sercu, T., Wadhawan, K., Padhi, I., Gehrmann, S., Cipcigan, F., Chenthamarakshan, V., Strobelt, H., Santos, C., Chen, P.y., Yang, Y.Y., Tan, J.P.K., Hedrick, J., Crain, J., Mojsilovic, A.: Accelerated antimicrobial discovery via deep generative models and molecular dynamics simulations. Nat. Biomed. Eng. (2021)
8. Dong, H.W., Hsiao, W.Y., Yang, L.C., Yang, Y.H.: Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment. AAAI Conf. Artif. Intell. pp. 34–41 (2018)
9. Duros, V., Grizou, J., Xuan, W., Hosni, Z., Long, D.L., Miras, H.N., Cronin, L.: Human versus Robots in the Discovery and Crystallization of Gigantic Polyoxometalates. Angew. Chemie - Int. Ed. **56**(36), 10815–10820 (2017)
10. Dutt, A.K., Müller, S.C.: Effect of stirring and temperature on the Belousov-Zhabotinskii reaction in a CSTR. J. Phys. Chem. **97**(39), 10059–10063 (1993)
11. Epstein, I.R., Pojman, J.A.: An introduction to nonlinear chemical dynamics: oscillations, waves, patterns, and chaos. Oxford University Press (1998)
12. Gao, W., Coley, C.W.: The Synthesizability of Molecules Proposed by Generative Models. J. Chem. Inf. Model. **60**(12), 5714–5723 (2020)

13. Gatys, L., Ecker, A., Bethge, M.: A Neural Algorithm of Artistic Style. J. Vis. **16**(12), 326 (2016)
14. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial networks. Adv. Neural Inf. Process. Syst. **27**, 2672–2680 (2014)
15. Granda, J.M., Donina, L., Dragone, V., Long, D.L., Cronin, L.: Controlling an organic synthesis robot with machine learning to search for new reactivity. Nature **559**(7714), 377–381 (2018)
16. Ha, D., Schmidhuber, J.: Recurrent world models facilitate policy evolution. Adv. Neural Inf. Process. Syst. **31**, 2451–2463 (2018)
17. Hsu, T.J., Mou, C., Lee, D.: Effects of Macromixing on the oregonator model of the belousov — zhabotinsky reaction in a stirred reactor. Chem. Eng. Sci. **49**(24), 5291–5305 (1994)
18. Lecun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-Based Learning Applied to Document Recognition. Proc. IEEE **86**(11), 2278–2324 (1998)
19. Lo, Y.C., Rensi, S.E., Torng, W., Altman, R.B.: Machine learning in chemoinformatics and drug discovery. Drug Discov. Today **23**(8), 1538–1546 (2018)
20. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.: Playing Atari with Deep Reinforcement Learning. arxiv.org **1312.5602** (2013)
21. Moen, E., Bannon, D., Kudo, T., Graf, W., Covert, M., Van Valen, D.: Deep learning for cellular image analysis. Nat. Methods **16**, 1233–1246 (2019)
22. Parrilla-Gutierrez, J.M., Hinkley, T., Taylor, J.W., Yanev, K., Cronin, L.: Evolution of oil droplets in a chemorobotic platform. Nat. Commun. **5**, 5571 (2014)
23. Parrilla-Gutiérrez, J.M., Sharma, A., Tsuda, S., Cooper, G.J., Aragon-Camarasa, G., Donkers, K., Cronin, L.: A programmable chemical computer with memory and pattern recognition. Nat. Commun. **11**, 1442 (2020)
24. Sanchez-Gonzalez, A., Godwin, J., Pfaff, T., Ying, R., Leskovec, J., Battaglia, P.W.: Learning to Simulate Complex Physics with Graph Networks. ICML (2020)
25. Shields, B.J., Stevens, J., Li, J., Parasram, M., Damani, F., Alvarado, J.I., Janey, J.M., Adams, R.P., Doyle, A.G.: Bayesian reaction optimization as a tool for chemical synthesis. Nature **590**(7844), 89–96 (2021)
26. Sterling, A.J., Zavitsanou, S., Ford, J., Duarte, F.: Selectivity in organocatalysis—From qualitative to quantitative predictive models. WIREs Comput. Mol. Sci. p. e1518 (2021)
27. 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. pp. 5999–6009 (2017)
28. Wang, T., Yuan, X., Trischler, A.: A joint model for question answering and question generation. arXiv **1706.01450** (2017)
29. Wiewel, S., Becher, M., Thuerey, N.: Latent Space Physics: Towards Learning the Temporal Evolution of Fluid Flow. Comput. Graph. Forum **38**(2), 71–82 (2019)
30. Xie, Y., Franz, E., Chu, M., Thuerey, N.: TempoGAN: A temporally coherent, volumetric GAN for super-resolution fluid flow. ACM Trans. Graph. **37**(4) (2018)