Deep Surrogate for Direct Time Fluid Dynamics

Lucas Meyer
EDF Lab Paris-Saclay, Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG
lucas.meyer@inria.fr

Louen Potier
EDF Lab Paris-Saclay, ENS Paris-Saclay
louen.pottier@ens-paris-saclay.fr

Alejandro Ribes
EDF Lab Paris-Saclay
91120 Palaiseau, France
alejandro.ribes@edf.fr

Bruno Raffin
Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG
38000 Grenoble, France
bruno.raffin@inria.fr

Abstract

The ubiquity of fluids in the physical world explains the need to accurately simulate their dynamics for many scientific and engineering applications. Traditionally, well established but resource intensive CFD solvers provide such simulations. The recent years have seen a surge of deep learning surrogate models substituting these solvers to alleviate the simulation process. Some approaches to build data-driven surrogates mimic the solver iterative process. They infer the next state of the fluid given its previous one. Others directly infer the state from time input. Approaches also differ in their management of the spatial information. Graph Neural Networks (GNN) can address the specificity of the irregular meshes commonly used in CFD simulations. In this article, we present our on-going work to design a novel direct time GNN architecture for irregular meshes. It consists of a succession of graphs of increasing size connected by spline convolutions. We test our architecture on the Von Kármán's vortex street benchmark. It achieves small generalization errors while mitigating error accumulation along the trajectory.

1 Introduction

Computational Fluid Dynamics (CFD) solvers have benefited from strong developments for decades, being critical for many scientific and industrial applications. Eulerian based CFD solvers rely on a discretization of the simulation space, i.e. a mesh, augmented with different fields like velocity and pressure, and constrained by initial and boundary conditions. The solver progresses by discrete time steps, building from the state u_t at time t, a new state $u_{t+\delta t}$ compliant with the Navier-Stokes equations. This process is compute intensive and often requires supercomputers for industrial grade simulations.

Connecting CFD with machine learning has received a renewed attention with the emergence of deep learning [23, 4]. The goal is often to augment or supplant classical solvers for improved performance in terms of compute speed, error, and resolution. In this paper, we focus on *Deep Surrogates* where a neural network is trained to provide a quality solution to the Navier-Stokes equations for a given domain, initial and boundary conditions.

Fourth Workshop on Machine Learning and the Physical Sciences (NeurIPS 2021).

Deep surrogates are currently addressed through different approaches. Data-free surrogates inject into the loss the different terms of the Navier-Stokes equations to comply with, leveraging automatic differentiation to compute the necessary derivatives. Data-driven surrogates train from the data produced by a traditional CFD solver. The surrogate can mimic the solver iterative process, being trained to compute $u_{t+\delta t}$ from u_t. But as the fluid trajectories are available at training time, other surrogates are trained to directly produce u_t from the parameters characterizing the conditions at t. Surrogates also differ in their approach to space discretization. If the mesh is a regular grid, CNNs can be used. Irregular meshes or particle based approaches are more challenging, and can be addressed through some variations of GNNs [3, 2]. These various approaches result in different trade-offs and limitations regarding precision, generalization capabilities, and scalability.

This paper presents our on-going work to design a surrogate for industrial and scientific CFD applications. The resulting surrogate is expected to be used for interactive data analysis, sensibility analysis, and digital twins. Precision and support for large irregular meshes are a priority rather than wide generalization capabilities as often targeted, for instance, by surrogates for computer graphics applications. Iterative surrogates tend to lose precision on long trajectories due to error accumulation. We thus opt for a direct time approach. But so far direct time surrogates have been mainly studied at small scale with regular meshes. Our contribution is a novel neural architecture that morphes the initial state parameters at t into the final irregular mesh providing u_t through a succession of layers of increasing size based on spline convolutions [9]. Early experiments with the Von Kármán's vortex street benchmark show that our architecture achieves small generalization errors (RMSE at about 10^{-2}) not subject to error accumulation along the trajectory.

2 Related Work

The seminal work of Raissi et al. introduces physics informed neural networks (PINNs) [19]. These data-free surrogate models are trained by minimizing the residual of the underlying Partial Differential Equation (PDE), whose terms are computed by automatic differentiation. PINNs have then been declined and applied to PDEs similar to the Navier-Stokes equations [12, 25, 6, 13]. In spite of their elegant data-free approach, PINNs are still unable to match traditional solver accuracy [5]. Whereas PINNs must learn representations of space and time to solve the PDE, other approaches guide further the model on how to treat space and time.

Iterative methods reconstruct the full dynamics by taking prediction u_t of the previous time step as input for computing the next one $u_{t+\delta t}$ [14, 21, 18, 25]. Inputs and outputs are not necessarily limited to one time step [27]. Data-driven iterative methods can only reproduce dynamics with the same time step δt as seen during training. Sanchez et al. overcome this limitation by mixing predictions of the derivative $u_{t+\delta t} - u_t$ and traditional numerical methods [20]. Liu et al. address this problem by training separately identical networks on different step sizes [16]. Iterative methods present a more serious problem: error accumulation. Each iteration of the surrogate model generates error, which accumulates along the simulation and eventually hinders its stability. In [21, 18], the authors add noise to the inputs to train the model to compensate its own error accumulation. Nonetheless, the initial noise level has to be set empirically. Traditional iterative solvers are also subject to error accumulation but with errors bounded by theoretical guarantees [10].

Instead of relying on intermediate steps, direct time prediction models predict u_t directly from the input time t. Consequently, the error is independent from other time steps. PINNs and their extensions provide examples of direct time predictions [19, 17]. The common approach, as found in [22], is to predict $u(x, t)$ from x and t representing a unique point located in space and time. Most of direct time prediction models are mesh-free. They do not take advantage of spatial correlations between inputs. To our knowledge, only [11] proposes direct time predictions on meshes. However, this surrogate model predicts steady flows, which by definition are flows that do not depend on time.

The meshes found in traditional solvers can also be used by surrogate models. In case they are regular, each time step can be seen as an image. This enables the use of traditional deep learning algorithms for image analysis tasks to predict u_t. For instance U-Net architectures are employed in [24, 26], or auto-encoders in [14]. At inference, these surrogate models are constrained by the size of the grid used during training. Conversely, models trained with irregular meshes adapt more easily to different resolutions. For instance, in [18] the network is trained with irregular meshes of coarse resolution and generalizes well to meshes of finer resolutions. Furthermore, irregular meshes are commonly

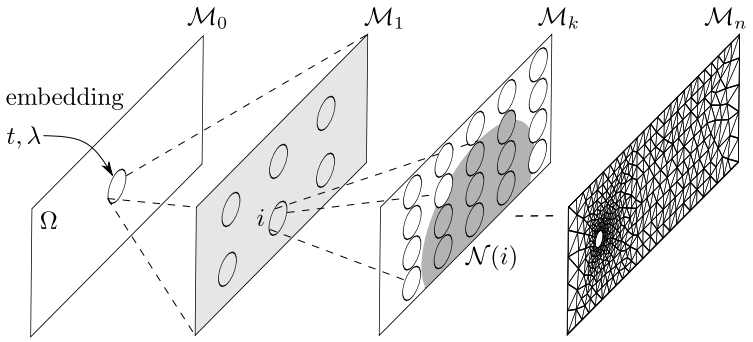


Figure 1: Architecture overview. A succession of regular meshes $\mathcal{M}_k$ defined over the domain Ω, starts with the singleton mesh $\mathcal{M}_0$ embedding the input values t, λ, and ends with the output $\mathcal{M}_n$ corresponding to the simulation mesh. Each node i contributes through convolutional operations to the nodes of mesh $\mathcal{M}_k$ that fall in its neighbourhood $\mathcal{N}(i)$.

used in CFD algorithms [7]. It is thus natural to find models trained with simulation data, such as [6], or mimicking traditional solvers [15], to be based on irregular meshes.

The long-time study of fluid dynamics provides physical knowledge that can be used as inductive bias to train a surrogate model. For instance, PINNs rely exclusively on the Navier-Stokes equations [19]. One critical element of these equations in the case of incompressible fluids is the incompressibility property itself. It translates as a divergence free velocity. To enforce this property is not trivial, even for traditional methods [7]. In [14] divergence free velocity fields are obtained by returning the curl of the network prediction. The incompressibility is also associated with pressure information traveling at infinite speed. In a graph based approach this means pressure information must be constantly shared between all the nodes, which would require as many message passing as the diameter of the graph. This would become quickly prohibitive for large meshes. Hence, [15] proposes a hierarchical network to let information flow across the whole mesh. We retain in our model this hierarchical approach to bring inductive bias.

3 Method

We propose a *data-driven direct time prediction deep surrogate model* (Fig. 1). Our deep neural network takes as input a time step t, and one or several physical parameters λ, which in the case of CFD represent fluid properties and/or initial conditions. The output of the network is a prediction $\hat{u}_t$, corresponding to the velocity and pressure fields of the fluid. This means that the network outputs all the values of a discretized field. Training is achieved with backprogation of the loss $\mathcal{L}$, which corresponds to the mean square error relative to the original simulation u_t: $\mathcal{L} = \|\hat{u}_t - u_t\|_2^2$.

The first key point in our approach is that the network is built by use of a mesh $\mathcal{M}(V, X)$, where V is the set of nodes of the mesh and X is the set of their spatial coordinates in the domain Ω. The mesh $\mathcal{M}$ corresponds to the spatial discretization used by the solver to generate the velocity and pressure fields. Thus, our deep surrogate outputs discretized fields (arrays of values) of the pressure and velocity, which are spatially localized by means of this mesh. Our aim is to integrate in the network architecture the geometrical information contained in $\mathcal{M}(V, X)$. For this, we build our network as a hierarchical graph neural network, i.e. a succession of graphs $\mathcal{M}_k(V^k, X^k, F^k)_{0 \leq k \leq n}$, where F^k denotes the set of feature vectors associated to the spatially localized nodes V^k. The last graph of the network $\mathcal{M}_n(V^n, X^n, F^n)$ is fixed by assigning $(V^n, X^n) \equiv (V, X)$ and F^n to the velocity and pressure field predictions. The preceding graphs $\mathcal{M}_{k<n}$ are defined as meshes of gradually thinner resolution starting from $\mathcal{M}_0(V^0, X^0, F^0)$, which is reduced to a singleton associated to an embedding of the inputs t and λ. The global architecture can be seen as a decoder that predicts the velocity and pressure from latent information about the time step and physical parameters, by means of progressively refined meshes.

The second key point is how consecutive graphs of our architecture are connected. We define a distance based neighborhood between meshes $\mathcal{M}_k$ and $\mathcal{M}_{k+1}$ as:

$$\forall j \in V^{k+1},\ \mathcal{N}(j) = \{i \in V^k, \|x_i - x_j\|_2 \leq r_k\}. \tag{1}$$

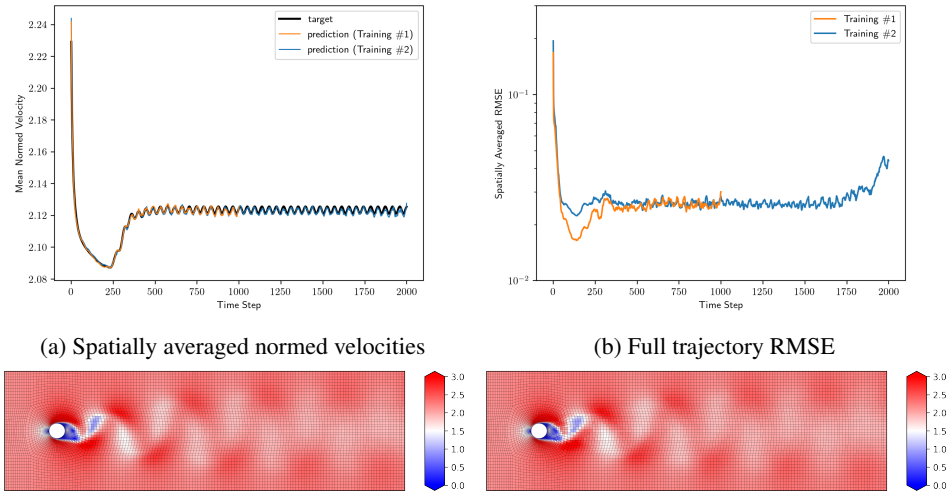
This reciprocally defines the neighborhood of any node $i \in V^k$ in V^{k+1} as displayed in Figure 1. Once this neighborhood is established, we build a non-regular convolution operator using the spline convolution defined in [9]. Thus we obtain a network allowing to deal with non-regular meshes of any dimension, which are the ones usually used in numerical simulations. Figure 1 highlights the hierarchical graphs used in the network architecture, which have a direct interpretation in the framework of graph neural networks [2]. The computation of features of $\mathcal{M}_{k+1}$ is achieved through a learnable node update function that takes as inputs the features of $\mathcal{M}_k$. Indeed, the feature vectors are computed as:

$$\forall j \in V^{k+1},\ \mathbf{f}_j = \frac{1}{|\mathcal{N}(j)|} \sum_{i \in \mathcal{N}(j)} \mathbf{H}_\theta(d_{ij})^\top \mathbf{f}_i \tag{2}$$

where the neighborhood $\mathcal{N}(j)$ is defined using Equation 1. The kernel $\mathbf{H}_\theta(d_{ij})$ is computed by interpolating the shifting vector d_{ij} between nodes i and j on m control points with B-splines functions of degree n along the d dimensions of Ω. Each basis function of the interpolation map is associated to a trainable weight. We refer the reader to the original article for a more detailed explanation about the spline convolution [9].

Our network consists in two fully connected layers that embed the parameters t and λ in a feature vector $\mathbf{f}_0$. Then a total of n spline convolution layers update nodes of $\mathcal{M}_{k+1}$ from values of $\mathcal{M}_k$. Each except the last convolution is followed by batch normalization and ReLU activation. In practice, we set $\mathcal{M}_{k<n}$ as regular meshes whose nodes are spread over Ω. We set $r_0 = +\infty$ and $r_k = \sqrt{2}h_k$, where h_k is the minimum distance between nodes of M_k. This allows each node of $\mathcal{M}_n$ to be parented to $\mathcal{M}_0$. The intuition is to share information between all the nodes of the output mesh to support the elliptic character of pressure information. The Navier-Stokes equations for an unsteady and incompressible fluid are incompletely parabolic with pressure information travelling at infinite speed across the mesh [7].

4 Experiment



(a) Spatially averaged normed velocities

(b) Full trajectory RMSE

(c) Normed velocity target at time step 1000

(d) Normed velocity prediction at time step 1000

Figure 2: Generalization test: (2a) compare the average normed velocity for prediction and target. (2b) displays the RMSE for the full trajectory. Training #1 is performed with the first 1000 time steps only, training #2 with 2000 time steps. (2c) and (2d) give a snapshot of target and prediction at time step 1000 for training #2.

We tested our approach on two canonical CFD examples, the lid-driven cavity and the von Kármán vortex street. For the sake of conciseness, here we only present results for the most complex

case. The von Kármán vortex street corresponds to a fluid passing a cylinder with imposed input velocity. Our architecture is implemented with pytorch-geometric [8], with 6 spline convolutions connecting 7 meshes of increasing resolution. The meshes are regularly covering the domain Ω with $h_{1\leq k\leq 6} = \lceil \frac{l}{2^{4-k}} \rceil$, where l is the length of Ω. The spline convolutions are of degree 1, with 3 points of control for each of the 2 dimensions. The numbers of output features for the 6 spline convolutions are respectively 512, 256, 128, 64, 32 and 3. The 3 final outputs correspond to the scalar pressure field and the 2-dimensional vector velocity field. The size of the network (34MB for 8.5M parameters) is considerably more compact than the training data. We used the Code-Saturne solver [1] to generate a dataset of 20 simulations (2GB per simulation). The 20 simulations ran with an irregular mesh of 7361 cells consisting of 15176 points, with an input velocity randomly sampled between 2 and 2.2 (Reynolds numbers between 3000 and 3300). Each simulation runs for 2000 time steps. The model inputs are thus $t \in [0, 2000]$ and $\lambda \in [2, 2.2]$. Train, validation, and test splits account respectively for 18, 1, and 1 simulations. Each simulation is normalized pointwise. We computed the mean and standard deviation of each point across the different time steps and set of parameters. Training took 1000 epochs using Adam optimizer. Initial learning rate was set to 5E-2 and decreased of factor 0.1 with a reduce on plateau scheduler of patience 10. The minimum learning rate was 1E-6.

Although never seen during training the test simulation belongs to the same distribution as the training dataset. Results shown here are thus interpolation both in time and velocity space. On the test simulation (Fig.2), the error averages 1.2E-2 with a standard deviation of 2.7E-2, similar to what was obtained during training. Generalization from training #2 well matches the target without an increasing error accumulation along the trajectory, except for the last time steps (above 1750). To further understand this phenomena, unexpected for a direct time approach, we performed a second training with only the 1000 first time steps (training #1). A similar increase of the error is visible when getting close to the last time steps. This suggests the observed increase is related to the lack of training data points surrounding the time horizon.

5 Conclusion

In this article we have proposed a novel deep surrogate architecture for Computational Fluid Dynamics. It differs from the state-of-the-art by its capability to compute the fluid state directly and to support irregular space discretizations. Early experiments show encouraging results with a generalization RMSE that is both low and not subject to error accumulation. So far, training and generalization were tested within a rather tight range of varying parameters. Future work will focus on testing and refining the presented architecture for a wider spectrum of simulation parameters.

References

[1] Frédéric Archambeau, Namane Méchitoua, and Marc Sakiz. "Code Saturne: A finite volume code for the computation of turbulent incompressible flows-Industrial applications". In: *International Journal on Finite Volumes* (2004).

[2] Peter W Battaglia et al. "Relational inductive biases, deep learning, and graph networks". In: *arXiv preprint arXiv:1806.01261* (2018).

[3] Michael M Bronstein et al. "Geometric deep learning: going beyond euclidean data". In: *IEEE Signal Processing Magazine* 34.4 (2017), pp. 18–42.

[4] Steven L Brunton, Bernd R Noack, and Petros Koumoutsakos. "Machine learning for fluid mechanics". In: *Annual Review of Fluid Mechanics* 52 (2020), pp. 477–508.

[5] Shengze Cai et al. "Physics-informed neural networks (PINNs) for fluid mechanics: A review". In: *arXiv preprint arXiv:2105.09506* (2021).

[6] Balthazar Donon et al. "Deep Statistical Solvers". In: *NeurIPS*. 2020.

[7] Joel H Ferziger, Milovan Perić, and Robert L Street. *Computational methods for fluid dynamics*. Vol. 3. Springer, 2002.

[8] Matthias Fey and Jan Eric Lenssen. "Fast graph representation learning with PyTorch Geometric". In: *arXiv preprint arXiv:1903.02428* (2019).

[9] Matthias Fey et al. "Splinecnn: Fast geometric deep learning with continuous b-spline kernels". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 869–877.

[10] Roland Glowinski. “Finite element methods for incompressible viscous flow”. In: *Handbook of numerical analysis* 9 (2003), pp. 3–1176.

[11] Lukas Harsch and Stefan Riedelbauch. “Direct Prediction of Steady-State Flow Fields in Meshed Domain with Graph Networks”. In: *arXiv preprint arXiv:2105.02575* (2021).

[12] Xiaowei Jin et al. “NSFnets (Navier-Stokes flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations”. In: *Journal of Computational Physics* 426 (2021), p. 109951.

[13] Ehsan Kharazmi, Zhongqiang Zhang, and George Em Karniadakis. “hp-VPINNs: Variational physics-informed neural networks with domain decomposition”. In: *Computer Methods in Applied Mechanics and Engineering* 374 (2021), p. 113547.

[14] Byungsoo Kim et al. “Deep fluids: A generative network for parameterized fluid simulations”. In: *Computer Graphics Forum*. Vol. 38. 2. Wiley Online Library. 2019, pp. 59–70.

[15] Mario Lino et al. “Simulating Continuum Mechanics with Multi-Scale Graph Neural Networks”. In: *arXiv preprint arXiv:2106.04900* (2021).

[16] Yuying Liu, J Nathan Kutz, and Steven L Brunton. “Hierarchical deep learning of multiscale differential equation time-steppers”. In: *arXiv preprint arXiv:2008.09768* (2020).

[17] Lu Lu, Pengzhan Jin, and George Em Karniadakis. “Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators”. In: *arXiv preprint arXiv:1910.03193* (2019).

[18] Tobias Pfaff et al. “Learning Mesh-Based Simulation with Graph Networks”. In: *International Conference on Learning Representations*. 2020.

[19] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. “Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations”. In: *Journal of Computational Physics* 378 (2019), pp. 686–707.

[20] Alvaro Sanchez-Gonzalez et al. “Hamiltonian graph networks with ode integrators”. In: *arXiv preprint arXiv:1909.12790* (2019).

[21] Alvaro Sanchez-Gonzalez et al. “Learning to simulate complex physics with graph networks”. In: *International Conference on Machine Learning*. PMLR. 2020, pp. 8459–8468.

[22] Justin Sirignano and Konstantinos Spiliopoulos. “DGM: A deep learning algorithm for solving partial differential equations”. In: *Journal of computational physics* 375 (2018), pp. 1339–1364.

[23] Rick Stevens et al. *AI for Science*. Tech. rep. Argonne National Lab.(ANL), Argonne, IL (United States), 2020.

[24] Nils Thuerey et al. “Deep learning methods for Reynolds-averaged Navier–Stokes simulations of airfoil flows”. In: *AIAA Journal* 58.1 (2020), pp. 25–36.

[25] Nils Wandel, Michael Weinmann, and Reinhard Klein. “Learning Incompressible Fluid Dynamics from Scratch–Towards Fast, Differentiable Fluid Models that Generalize”. In: *arXiv preprint arXiv:2006.08762* (2020).

[26] Rui Wang et al. “Towards physics-informed deep learning for turbulent flow prediction”. In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2020, pp. 1457–1466.

[27] Steffen Wiewel, Moritz Becher, and Nils Thuerey. “Latent space physics: Towards learning the temporal evolution of fluid flow”. In: *Computer graphics forum*. Vol. 38. 2. Wiley Online Library. 2019, pp. 71–82.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper’s contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] "So far training and generalization were tested within rather tight range of varying parameters" in Section 5
   (c) Did you discuss any potential negative societal impacts of your work? [No] The most obvious negative impact of our work would be blind application of the model to simulate critical engineering simulation. The model however is far from being mature enough to be used in such context.

(d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]

2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A] We didn't include theoretical results.
   (b) Did you include complete proofs of all theoretical results? [N/A] We didn't include theoretical results.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No] The work is still on-going, but data and code can be shared later upon request.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section 4
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Section 4
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] We used and cited open source Code_Saturne solver[1] to generate the data and pytorch-geometric[8] to build the model.
   (b) Did you mention the license of the assets? [No]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A] We didn't use crowdsourcing or conduct research with human subjects.
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A] We didn't use crowdsourcing or conduct research with human subjects.
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A] We didn't use crowdsourcing or conduct research with human subjects.