**Machine Learning in Neuronal Modeling: A Comparative Analysis and Implementation**

**CMSE 410 – Final Project Report**



Jaini Gandhi

Instructor: Prof Jianrong Wang

*Problem and Goals:*

Understanding and predicting how neurons respond to stimuli is a central problem in computational neuroscience. Classic neuron models like the Hodgkin-Huxley [1],[2] equations and integrate and fire models [3] provide mechanistic insight by using differential equations to reproduce action potentials. For example, the seminal work of Hodgkin and Huxley (1952) established how ion currents produce spikes in the squid giant axon. Such biophysical models are powerful but require extensive parameter tuning and are computationally intensive​. This limits their scalability to large networks.

Neurons communicate using sequences of action potentials. The number of spikes fired in a response to an input is a simple but informative summary of neural activity, closely related to firing rate. In many neural coding theories, spike count or firing rate conveys information about stimulus intensity or other features. For instance, stronger current injections generally evoke more spikes, reflecting how neurons encode input magnitude in their output rate. Thus, being able to predict the spike count from a given stimulus is important for understanding neuronal input-output relationships. If an ANN could accurately emulate a neuron's spike count response, it would pave the way for replacing hand-crafted neuron models with learned models in brain simulations. This motivation aligns with broader efforts to integrate machine learning and neuroscience, as seen in work on multi-scale brain network modeling and on spiking neural networks bridging biology and AI [5].

I chose this project to explore if modern machine learning, specifically artificial neural networks, could efficiently predict neuron behavior, such as spike counts in response to stimuli, providing a faster alternative to biophysical models. Initially, my goal was broad: compare several ML models (ANNs, LSTMs and SNNs) for predicting neuron responses, focusing on efficiency and biological plausibility. By midterm, the scope narrowed to focus on a straightforward feedforward ANN to predict spike counts from stimulus features. SNNs while biologically realistic, are complex to implement and train and LSTMs are powerful for sequence modeling but likely overkill for predicting an aggregate count from a fixed duration stimulus. Therefore, due to their simplicity and proven efficiency on similar regression tasks, ANNs became my primary choice. Thus, my project shifted to asking: Can an ANN accurately predict the number of spikes a neuron fires based on initial stimulus features? Not only did I want to assess if an ANN could make accurate predictions, but I also sought to understand why it might struggle and what could make future implementations successful.

In summary, the project’s driving goal is to explore data-driven spike count modeling as a step toward efficient yet biologically relevant neuron models, while acknowledging the challenges in capturing neuron dynamics with machine learning.

*Datasets:*

We used the Allen Institute’s Brain Atlas Cell Types Database as our data source [6]. This open-access resource provides a wealth of electrophysiological recordings from individual neurons, obtained via standardized whole-cell patch clamp experiments [9]​. Specifically, we drew from the human cortical cell dataset (adult human brain tissue) since it offered a large number of cells (over 400 neurons) with consistent recording protocols. Each recording consists of multiple sweeps where a defined current was injected, and the neuron’s voltage response was captured. For consistency, I chose sweeps that used a 1-second square current stimulus, as they reliably evoke measurable spike counts.

The data was in the Neurodata Without Borders (NWB) format, which organizes the stimuli, voltage responses and the metadata in a standardized HDF5 based file. Data was accessed using AllenSDK in Python. The raw data for a given neuron includes voltage responses to various injected current stimuli, as well as metadata about each stimulus “sweep” (stimulus type, amplitude, duration, etc.) [10]. The data format met the expectations in that it was high quality and fairly well-documented. However, working with it still required careful preprocessing.

For each neuron, multiple sweeps are recorded. In the subset, many sweeps were step-current injections of different amplitudes (including positive current steps of varying magnitude, and some negative or zero-current steps for baseline). The typical protocol delivered a current step starting at 1 second into the recording and lasting ~1 second. Our target output for each sweep was defined as the total number of action potentials (spikes) fired by the neuron during the 1-second stimulus window. I used the AllenSDK’s built-in utilities to detect spikes from the voltage trace, which provided a robust way to count spikes for each sweep. As expected, sweeps with larger current amplitudes generally evoked more spikes, whereas sweeps with very small or negative current (hyperpolarizing stimuli) produced zero spikes. I decided to ignore or filter out cases with negative or zero current input for modeling purposes, since those trivially result in zero spikes and would have dominated the training data with zeros. After filtering, a dataset of approximately 25,000 samples (sweeps) from 413 human neurons was complied, each sample characterized by the input stimulus and the resulting spike count.

Each sample in the processed dataset has associated features. I computed summary features of the stimulus, such as: mean current, standard deviation of the current, the integral of the current over time, and the stimulus duration. A feature for the neuron’s baseline voltage noise was also included, calculated as the voltage standard deviation just before the stimulus onset – this was to capture any irregularity or spontaneous activity in the neuron prior to stimulation. These features formed the input vector for the initial ANN model. In addition, I later engineered a few combined features (energy = integral², complexity = std/mean, power = integral/duration) to provide the model with some nonlinear combinations that might be informative. The output for each sample is simply an integer spike count.

By midterm, I had successfully gathered and organized the data as described. The dataset largely met the needs: it was rich and diverse, containing neurons with a wide range of excitability. One minor surprise was the degree of variability across cells – some neurons would fire 20+ spikes at high current, while others might only fire 5 at the same current, or even none at all if they were low-excitability cell types. This heterogeneity became a challenge for the ANN to learn, as discussed later. I did not make major changes to the dataset after midterm, but I did perform additional cleaning such as removing a few outlier sweeps where spike detection failed or where stimuli were unusual.

*Computational Approach:*

*A screenshot of a computer

Description automatically generated*The computational approach followed a typical data-driven modeling workflow, which is shown schematically in Figure 1 (workflow diagram). In brief, the steps were: (1) Data acquisition and preprocessing, (2) Feature extraction, (3) Model training, and (4) Evaluation. Below each step is documented in detail, including what was attempted at each stage between midterm and final, the outcomes, and insights gained.

Data Acquisition and Preprocessing: Using the AllenSDK, electrophysiology data was programmatically fetched for each neuron. This involved iterating over all human neuron IDs and downloading their NWB files (about 413 cells). For each neuron, model iterated through all sweeps. I focused only on sweeps where the stimulus was measured in Amps (i.e. actual current injections, skipping things like voltage clamp or ramp stimuli identified by different units). For each sweep, the time-series of stimulus current and membrane voltage was loaded. Then the 1-second window during the stimulus were then isolated and this segment was resampled to a fixed length (10,000 time points at 10 kHz) to normalize all inputs. Using the AllenSDK’s EphysSweepFeatureExtractor, I detected spike times within that window and counted them. This yielded the ground-truth spike count for the sample​. Each sample’s data (stimulus and response) was appended to the dataset arrays. By the end of this process, I had a large NumPy array X of shape (~25k samples × 10000 timepoints × 2 channels), and a vector y of spike counts. An immediate insight from this preprocessing was that the distribution of spike counts was highly skewed – many sweeps (especially those with weak stimuli) had 0 spikes, and relatively fewer sweeps had large spike counts.

*Figure 1: Flowchart showing the steps taken to approach the problem computationally.*

Feature Extraction: In parallel with the above, I computed summary features for each sweep to use in a simpler model. These included the mean current which for step stimuli corresponds to the amplitude, the standard deviation of the current, the integral of the current which in the case of a step is just mean × duration, the duration, and baseline voltage standard deviation prior to stimulus. These were stored in a data frame per sample. After midterm, I engineered a few additional features from these basics – for example, stimulus energy (the squared integral, highlighting differences between short high-current pulses vs. longer lower-current injections delivering the same total charge), stimulus complexity (defined as standard deviation/mean, essentially the coefficient of variation of the stimulus waveform, to distinguish constant vs. varying stimuli), and stimulus power (integral divided by duration, which for the fixed window is proportional to mean, but would differ if some stimuli ended early). The motivation was to see if these derived features could help the model pick up nonlinear relationships. At this stage (midterm), some exploratory analysis was performed. Indeed, as expected, spike count correlates strongly with stimulus amplitude at the level of individual neurons. For example, in one neuron, a 50 pA current might yield 0 spikes, 100 pA yields 1 spike, 150 pA yields ~5 spikes, and higher currents saturate around that neuron’s maximum firing rate (Figure 2). Across different neurons, however, the curves shifted – some neurons required >100 pA to start spiking, while others would spike with only 50 pA. This variability suggested that a global model would have to either learn each neuron’s threshold or somehow account for neuron-specific differences (which our initial feature set did not explicitly encode). I considered adding the neuron’s identity as a one-hot feature vector, but with 400+ neurons, that would be impractical and risk overfitting. Thus, I proceeded with the features at hand, acknowledging that the model might effectively be averaging over heterogeneous cells.

(Figure 2 here — Spike Count vs. Stimulus Amplitude plot for an example neuron. This plot shows how the number of spikes fired increases with the injected current. The relationship is roughly sigmoidal: low currents produce 0 spikes, then beyond a threshold the neuron begins to fire, and eventually plateaus at a maximum firing rate. Each point represents a sweep at a particular current amplitude.)

Baseline ANN Model: My first modeling attempt was a simple feedforward ANN that takes the extracted features as input and predicts the spike count. I treated it as a regression problem. I built a network in TensorFlow/Keras with an input layer of size equal to the number of features (8 features were used: the five basic ones plus the three engineered combos). I included three hidden layers with decreasing size (128, 64, and 32 neurons respectively), each with ReLU activations, and inserted Dropout (20–30% rate) after each hidden layer to reduce overfitting. The output layer was a single neuron with linear activation (for predicting a continuous value, the log spike count). I trained this model using the Adam optimizer to minimize Mean Squared Error (MSE) loss. We also monitored Mean Absolute Error (MAE) for interpretability (MAE in spike count units) and used early stopping on validation loss to prevent wasteful epochs. The data was split into training and test sets (75/25 split). Before training, all input features were standardized (zero mean, unit variance) to help network convergence. The spike count was also log-transformed as noted, meaning the network actually learned to predict log(count), this helped stabilize the loss because an error of say 10 spikes (when actual is 50 vs. 60) is proportionally smaller than an error of 10 when actual is 10 vs. 20; the log-transform down-weights large differences.

Training the ANN was fast, however, the training loss decreased and converged within ~20 epochs, at which point validation loss stopped improving. Figure 3 shows the training and validation loss curves for this model. The validation MSE plateaued around a value that corresponded to an MAE of about 2 spikes. For context, the average spike count in the data was around 2–3, so an error of 2 is fairly large. Indeed, when the model was evaluated on the test set, the results were underwhelming: an R2 (coefficient of determination) essentially -0.03 (the model did *worse* than predicting a constant average in terms of variance explained). In practical terms, the model often predicted a small number of spikes (maybe 0–3) for most inputs, missing high-count cases and sometimes overestimating low-count ones. The scatter plot of predicted vs. actual spike counts (Figure 4) for the test set illustrates this – if the model were perfect, all points would align on the diagonal, but the points were widely scattered. For instance, there were sweeps where the neuron actually fired 10+ spikes, but the model predicted only 4 or 5, and vice versa. The model did capture the trivial cases (when stimulus was very low, it correctly predicted 0 spikes, and when stimulus was very high, it tended to predict a higher count), but it failed to account for the nuances between different neurons. I interpreted this as the model underfitting the data: the simple feature set and network were not expressive enough to learn the complex, neuron-specific relationships. Another factor is that one model was trained across all neurons without giving it a way to know *which* neuron a sample came from. Essentially, the model learned an “average neuron’s” response to current, which is not a great predictor for any given cell. This was a key insight: to improve, I’d need either to incorporate neuron identity or train separate models or add more informative features.

Advanced ANN Model: After the baseline model, I attempted a more sophisticated approach using the raw time-series data (stimulus and voltage trace) as input to an ANN. The intuition was that a deep network might be able to infer each neuron’s responsiveness dynamically from the shape of its voltage trace. For example, a neuron’s voltage response in the first 50 ms of a current pulse could reveal its excitability (how quickly it starts to depolarize or fire), which could allow the network to predict how many total spikes will occur by the end of the 1-second pulse. I designed a 1D CNN that takes a two-channel sequence (current *I* and voltage *V* over time) and outputs the spike count. The CNN architecture had two convolutional layers: the first with 32 filters, the second with 64 filters, both with small kernel size (5) and ReLU activation, each followed by a max-pooling layer (down sampling by factor of 2) and dropout (20%). These convolutional layers should act as feature extractors, potentially detecting spike waveforms or subthreshold oscillations in the voltage channel, or particular stimulus patterns in the current channel. After flattening the conv outputs, we added a dense layer of 64 neurons with ReLU and dropout (30%), then the final output neuron. The CNN was also trained with the same loss (MSE on log spike count) and optimizer.

Training the CNN was computationally intensive. With ~20k training samples of length 10k, and a fairly deep network, each epoch took several minutes on our machine. I initially set an epoch limit of 100 with early stopping, but in practice I had to stop training after ~5 epochs due to time. The training loss did decrease substantially in the first epoch or two (starting from a high value since weights were random), but then it began to fluctuate. By epoch 3–4, the model seemed to be struggling to continue improving validation loss – in fact, I observed possible overfitting as the training loss kept decreasing while validation loss started rising slightly after a point. This was a sign that our CNN might have had more capacity than the data could robustly train, or simply that we hadn’t trained long enough to reach a better minimum. We suspect that with more epochs (and maybe some regularization tweaks), the CNN could eventually learn to count spikes by detecting them in the voltage trace. In partial training, we already saw that the CNN’s predictions correlated with the presence of spikes: it almost never predicted a high count when there were zero spikes, and vice versa. But it wasn’t very accurate beyond that qualitative alignment. Due to time constraints, I did not pursue the CNN further and treated just the fully connected ANN (with features) as my main result for analysis, since it was fully trained and easier to interpret.

*(Figure 3 here — Training and Validation Loss curves for the fully connected ANN model. The plot shows MSE loss over epochs. The training loss (blue curve) decreases steadily, while the validation loss (orange curve) levels off early, indicating that additional training beyond ~20 epochs doesn’t improve generalization. This suggests the model has learned whatever structure it can from the given features, and further training risks overfitting.)*

*(Figure 4 here — Predicted vs. Actual Spike Count scatter plot for the test set (ANN model). Each point represents one test sweep. The x-axis is the actual spike count (ground truth from data), and the y-axis is the ANN’s predicted spike count. The red dashed line is the ideal $y=x$ line. Our model’s points show a lot of spread around the ideal line, with a slight tendency to under-predict higher spike counts. For many zero-spike cases the model correctly predicts zero (points on the origin), but for higher counts the predictions are often significantly off. This underscores the model’s limited accuracy.)*

*Key Findings:*

Despite the challenges, the project yielded several important findings and takeaways:

Model Performance: The ANN was able to learn the general trend that higher input current leads to more spikes, but it largely failed to capture neuron-specific firing rates. Quantitatively, the fully trained model’s predictions explained essentially 0% of the variance in spike counts on held out data. The MAE was ~2 sikes, which in many cases in on the order of the actual spike count itself. Alternatively, if a sweep had 4 spikes, a typical prediction might be 2 or 6 – a big relative error. The CNN model did not significantly outperform the simpler ANN in the limited training we managed. These negative results highlight that our approach, as implemented, underfit the complexity of the problem. However, it’s worth noting that the model did get the trivial cases right (zero spikes when no current) and never produced physically impossible predictions (negative spike counts).

Data Insights: Through this process, the firing rate (spike count) vs. current relationship was found to be monotonic and roughly sigmoidal for individual neurons, consistent with basic neuroscience knowledge about f-I curves. The data showed clear firing thresholds and saturation points for each cell. It was also easy to see how diverse neuronal responses can be: some neurons were much more excitable than others. This biological variance is not just “noise” – it carries meaning (different cell types or states) – and my attempt to have a single model account for all of it was probably overambitious.

ANN vs. Mechanistic Expectations: One initial question was whether a generic ANN could match a biophysical model in predicting spikes. While I wasn’t able to fully implement a biophysical model, based on literature I can contextualize our results. A previous study [11] successfully trained ANNs to predict spike occurrences in simulated Hodgkin-Huxley neurons with high accuracy​. In our case with real neurons, the ANN’s low accuracy suggests that either the model was too simple, or the real data had more variability than a simulation. It’s likely a bit of both. Real neurons are not identical copies of each other, and factors like experimental noise or cell health can affect firing. The findings imply that to get ANNs to perform well on real neural data, one might need either more complex networks or a strategy to handle cell-specific factors (which the mechanistic models inherently do by fitting parameters per cell). On a positive note, I demonstrated a complete pipeline that can be iterated upon.

In summary, the key result is that a simple ANN trained on summary features failed to reliably predict spike counts across a population of human neurons. This negative result is itself informative: it points to the necessity of incorporating more information (about individual neurons or temporal dynamics) for success. It was also found that training on raw waveforms is computationally heavy and didn’t yield immediate benefits, suggesting that smarter architectures or more training time would be needed for that approach. Overall, I have a working framework that can be incrementally improved and has taught me about both the data and the modeling problem.

Code and Data Availability: All code used for preprocessing, model training, and analysis, as well as the processed dataset (feature tables and spike counts), are available in the project GitHub repository (GitHub link *[placeholder]*). This includes Jupyter notebooks demonstrating data extraction with AllenSDK, feature engineering, and model training/evaluation. The original raw data can be obtained from the Allen Cell Types Database website using the provided cell IDs in our repository.

*Challenges:*

This project was not without its challenges. One of the biggest challenges was scope management - my initial goals were quite broad, and I had to trim them down significantly. For example, I hoped to implement a biophysically inspired model (like a simple neuron model) for comparison but integrating that in the pipeline proved too time-consuming. Instead, I focused on the data-driven ANN part and treated the literature as my comparative benchmark. In hindsight, trying to do both within one semester was overly ambitious.

Another challenge was dealing with the heterogeneity of the biological data. As discussed, the variability between neurons meant that the model was always chasing a moving target. I didn’t anticipate just how much this would impact training. When the model started averaging out responses, it took a moment to realize that it wasn’t simply failing – it was finding the best compromise given conflicting patterns. In practical terms, one manifestation was that the training loss would decrease to a point but the validation loss wouldn’t budge, hinting that the model capacity was going to learning an average solution. I tried adding complexity (the CNN) to let it possibly differentiate patterns, but then we hit the next challenge: computational resources.

Training the CNN on 10,000-step sequences for ~25k samples was computationally intensive. I had limited GPU access, so we mostly trained on CPU which was slow. We had to cut short the CNN training to a few epochs due to time. This prevented us from fully seeing its potential. It was a bit frustrating to have a complex model that I suspected could work better, but not being able to properly tune or run it to convergence. This taught us about the importance of computational budgeting and maybe using a smaller dataset or shorter signals for prototyping.

Logistics-wise, working with the AllenSDK and the large data files required some patience. I had to ensure I cached data to disk to not repeatedly download, manage memory when building big arrays, etc. Minor issues like data type mismatches (units in A vs. pA) could have thrown off results. These issues were resolved through careful reading of documentation and trial and error.

Lastly, a subtle challenge was time management and balancing project elements. I spent a lot of time on data preparation (which was necessary), but it left less time for model experimentation. In a sense, I perhaps underestimated how long the data wrangling would take, which is a common situation in data science. By the time I had everything ready, we had to rush some of the modeling, which is why some advanced ideas (like hyperparameter tuning, or trying an RNN) were left on the table.

In summary, the challenges faced were both technical (data processing, computational limits) and conceptual (model underfitting, biological variability). Each challenge taught me something: e.g., a negative result isn’t a failure if it reveals a truth about the problem, and cutting down scope is necessary to make meaningful progress.

*Reflection and Future Directions:*

If I were to restart this project knowing what I know now, I would do a few things differently. First, I would consider narrowing the scope further to make the modeling task more tractable. For example, I might choose a subset of neurons that are more homogeneous (say, only excitatory neurons from a certain cortical layer) so that the input-output relationship is more consistent. This could dramatically improve the model’s performance and allow us to then incrementally add complexity (like including more neuron types) to see how it copes. By focusing on one cell type at a time, the ANN could effectively become a specialist, and I could later create an ensemble or multi-headed model for multiple types.

Second, I would reconsider my choice of model architecture. A fully connected network on summary features turned out to be too limited, while a CNN on raw data was maybe too ambitious given resources. A middle ground could be using a recurrent neural network (RNN) or a temporal convolution on the stimulus time-series only. Essentially, treat the neuron as an unknown dynamical system and have an RNN try to emulate it. The output could be not just final spike count, but even a sequence of instantaneous firing probability that we integrate or threshold. I avoided sequence-to-sequence or sequence-to-event modeling due to complexity, but a future direction could revisit it. There are also spiking neural network models or differentiable event counters that could be applied for a more neuroscience-flavored approach.

Third, more extensive feature engineering or data augmentation could help. For instance, I could simulate a small amount of data using known neuron models to augment the training – e.g. generate some artificial neurons that have known parameters and include them in training to guide the ANN. Or perform transformations on the existing data. Another idea is to incorporate the stimulus waveform shape more directly: since many sweeps were just steps, the features I had were enough, but for those that were stimuli with transients, maybe more descriptive features (like frequency content, if it was a noisy stimulus) would help.

I was also interested in bridging back to mechanistic models. A future direction is to fit a simple LIF (leaky integrate-and-fire) model to each neuron’s data and use those fitted parameters as input to an ANN. Essentially, tell the ANN each neuron’s “capacitor and resistor values” so it has an idea of the neuron’s time constant and excitability. This could combine interpretability and learning nicely – the ANN wouldn’t be fully a black box because it uses meaningful descriptors.

In future iterations, I would also devote more effort to hyperparameter tuning (e.g. learning rate schedules, different activation functions). Perhaps the use of a different loss function could help – for example, a Poisson regression loss might be more appropriate if I consider spike count as a Poisson-distributed variable given the stimulus. Treating it as a regular squared error problem might not be statistically optimal. This is an interesting avenue merging statistical modeling with ANN training (e.g. using a negative log-likelihood for count data).

In summary, there are many ways to improve and extend this work. I learned a lot from what didn’t work, and that knowledge will inform future attempts. Key future directions include incorporating neuron-specific information, exploring more suitable model architectures (like RNNs or mixture models), and perhaps focusing the task on more homogenous data or a per-neuron basis. With these changes, I expect significantly better performance in predicting spike counts – and maybe even extending to predicting the timing of spikes, which would be the ultimate goal.

*Acknowledgements:*

I would like to express my gratitude to those who supported this project. Prof. Jianrong Wang provided invaluable guidance and feedback throughout the project. I also thank the Allen Institute for Brain Science for making the Cell Types Database openly available; such resources are the backbone of computational neuroscience research. The Allen SDK and documentation were immensely helpful for getting started with the data – and the community forums clarified many usage questions. Additionally, I’m grateful to my friends and family for their patience and moral support. A special thanks to a colleague who helped me debug early code and shared tips on using H5PY for NWB files.

*References*

1. Hodgkin, A. L., Huxley, A. F., & Katz, B. (1952). Measurement of current-voltage relations in the membrane of the giant axon of Loligo. The Journal of Physiology, 116(4), 424–448.
2. Johnson, M. G., & Chartier, S. (2017). Spike neural models, part I: The Hodgkin-Huxley model. The Quantitative Methods for Psychology, 13(2), 105–119.
3. Izhikevich, E. M. (2003). Simple model of spiking neurons. IEEE Transactions on neural networks, 14(6), 1569-1572.
4. Schirner, M., McIntosh, A. R., Jirsa, V., Deco, G., & Ritter, P. (2018). Inferring multi-scale neural mechanisms with brain network modelling. eLife, 7, e28927.
5. Yamazaki, K., Vo-Ho, V. K., Bulsara, D., & Le, N. (2022). Spiking neural networks and their applications: A review. Brain Sciences, 12(7), 863.
6. Allen Institute. Cell Types Database – Overview of the Data. Allen Brain Atlas: Cell Types. (n.d.). Retrieved from <https://celltypes.brain-map.org/data>.
7. Deng, L., Wu, Y., Hu, X., Liang, L., Ding, Y., Li, G., ... & Xie, Y. (2020). Rethinking the performance comparison between SNNs and ANNs. Neural Networks, 121, 294–307.
8. Souza de Abreu, R., Silva, I., Nunes, Y. T., Moioli, R. C., & Guedes, L. A. (2023). Advancing fault prediction: A comparative study between LSTM and spiking neural networks. Processes, 11(9), 2772.
9. Gouwens, N. W., Sorensen, S. A., Berg, J., Lee, C., Jarsky, T., Ting, J., ... & Koch, C. (2019). Classification of electrophysiological and morphological neuron types in the mouse visual cortex. Nature neuroscience, 22(7), 1182-1195.
10. Lee, B. R., Budzillo, A., Hadley, K., Miller, J. A., Jarsky, T., Baker, K., ... & Berg, J. (2021). Scaled, high fidelity electrophysiological, morphological, and transcriptomic cell characterization. Elife, 10, e65482.
11. Wang, T., Wang, Y., Shen, J., Wang, L., & Cao, L. (2022). Predicting spike features of hodgkin-huxley-type neurons with simple artificial neural network. Frontiers in computational neuroscience, 15, 800875.