Performance analysis of grid cells for temporal data predictions.

Kateri Duranceau,Taahaa Mir*
Department of Computer Science
McGill University
845 Sherbrooke St W, Montreal, Quebec H3A 0G4, Canada
{taahaa.mir,kateri.duranceau}@mail.mcgill.ca

Isabeau Prémont-Schwarz†
Department of Computer Science
McGill University
845 Sherbrooke St W, Montreal, Quebec H3A 0G4, Canada
isabeau.premont-schwarz@mcgill.ca

Abstract

This paper investigates the performance of one dimensional grid cells for temporal data predictions. Utilizing the Vector-HaSH content addressable memory model, we implemented a single dimensional grid cell configuration, and implemented a modified recall process to make forecast predictions given previous temporal data that our model stored in memory. We track the performance of our model's predictions using the M4 forecast competition benchmarks[2] and compare our results to the results obtained by other models in the M4 competition. Our episodic memory recall model, the Grid4Cast, is made of four different prediction algorithms. Results show that Grid4Cast performs well on daily and yearly data, achieving low error scores and strong predictive accuracy, particularly for long-term trends. However, the model faced challenges with hourly, weekly, and monthly data, where noise and irregular patterns reduced performance. Our findings display the potential in using biologically inspired models for forecast prediction while also highlighting the need for finer tuning of these models.

1 Introduction

Episodic memory, a foundation of human cognition, enables the encoding, storage, and retrieval of specific events contextualized by time and space. This memory mechanism is characterized by its ability to store sparse, temporally structured patterns and recall them in response to incomplete or noisy cues. Inspired by these biological processes, computational models such as the Vector-HaSH Content Addressable Memory (CAM) network have emerged, offering efficient methods for encoding high-dimensional patterns and enabling associative recall. These networks are particularly well-suited for handling sequential data, providing a robust framework for tasks requiring temporal continuity.

In parallel, time series forecasting—a critical task across domains such as finance, healthcare, and climate modeling—relies on recognizing and extrapolating patterns from historical data. However, traditional forecasting models often struggle to balance noise robustness and long-term temporal dependencies. Integrating the principles of episodic memory with the retrieval capabilities of Vector-

*Code is available at: https://github.com/tmir00/TemporalNeuroAI

†Corresponding author

Preprint. Under review.

HaSH CAM presents a novel approach to enhance time series prediction. Using sparse representations and associative retrieval, such systems can capture and generalize temporal patterns with improved accuracy and efficiency.

This work explores the intersection of episodic memory, time series forecasting, and Vector-HaSH CAM networks. We propose a framework, the Grid4Cast model, that mimics the memory processes observed in biological systems, applying these principles to enhance predictive modeling in time series analysis.

2 Related Works

Our work was significantly inspired by two experimental models that leveraged the biological phenomenon of grid cells, the Memory Scaffold with Heteroassociation (MESH) model [4] and the Vector Hippocampal Scaffolded Heteroassociative Memory (Vector-HaSH) model [1]. The Vector-HaSH framework is a continuation of the MESH framework, improving its original Content-addressable Memory (CAM) network.

2.1 MESH: Memory Scaffold with Heteroassociation

This paper introduces a novel content-addressable memory (CAM) architecture called Memory Scaffold with Heteroassociation (MESH), designed to overcome the limitations of existing CAM networks, particularly the *memory cliff*—a point at which adding a single additional pattern causes the catastrophic loss of all stored patterns. Inspired by the Entorhinal-Hippocampal memory system in the brain, MESH separates memory storage into two components: a *memory scaffold* of predefined, stabilized states, and a *heteroassociative* process that links these states to external patterns [4].

This architecture creates a CAM continuum that avoids the memory cliff, allowing for near-perfect recall of small numbers of information-dense patterns and partial recall as more patterns are stored, with a graceful trade-off between the number of patterns and the richness of information per pattern. MESH achieves near-optimal information storage for any number of patterns, outperforming traditional CAM models [4]. The paper also discusses the theoretical foundation of CAM networks, presents central results on MESH's performance, and extends the model to continuous neural activations, demonstrating its robustness across different types of data.

2.2 Vector-HaSH: Vector Hippocampal Scaffolded Heteroassociative Memory

The Vector-HaSH model proposes a neocortical-entorhinal-hippocampal network that supports high-capacity associative memory, spatial memory, and episodic memory [1]. The model leverages pre-structured representations (grid cells) interacting with hippocampal cells through a scaffold of fixed and random projections. This structure allows for the creation of stable fixed points, enabling high capacity and robust memory storage and recall. Key findings include the model's ability to generalize from a small set of learned states, high-capacity storage without catastrophic forgetting, and accurate memory recall even with partial inputs.

The Vector-HaSH model is designed to explain how the hippocampus stores and recalls memories, involving multiple layers that work together to create and maintain stable memory representations. These layers include the Input Layer, the Grid Cell Layer, the Hippocampal Cell Layer, and the Scaffold, which encompasses the connections and interactions between these layers (as shown in figure 1).

Traditional models of hippocampal memory, such as Hopfield networks, have limitations in capacity and susceptibility to the memory cliff problem. Recent advances have introduced more complex models integrating spatial representation and episodic memory. The Vector-HaSH model stands out by separating the dynamics of memory storage from content, utilizing a scaffold to integrate grid and hippocampal cells [1]. Compared to previous models, Vector-HaSH demonstrates superior generalization and resilience to interference.

Furthermore, Vector-HaSH offers several advantages over the MESH architecture. While MESH provided a high-capacity memory with a trade-off between the number of stored patterns and recall fidelity [4], it had limitations, such as not requiring grid cell encodings, weak generalization in scaffold learning, and lower sequence capacity. In contrast, Vector-HaSH utilizes a memory scaffold

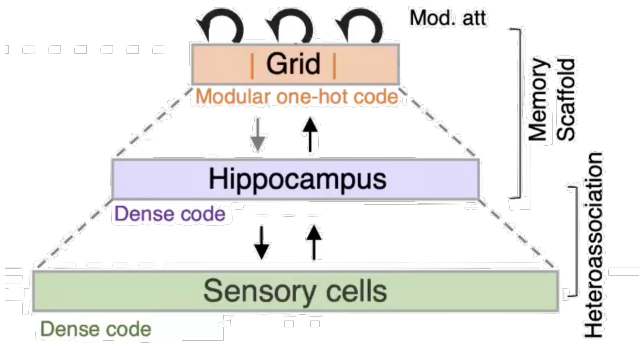


Figure 1: Layers in the Vector-HaSH Structure

based on a recurrent circuit incorporating MEC (medial entorhinal cortex) grid cells and hippocampal layers, which enhances sequence capacity and ensures strong generalization. The grid cells in Vector-HaSH are represented using a one-hot encoded vector on a discretized hexagonal lattice, leading to more robust and accurate memory encoding. This architecture allows for improved pattern recall and sequence learning, making it a superior option compared to MESH [1].

3 Methodology

In this section, we detail the implementation and evaluation of the 1D grid cells within the Grid4Cast model, describing the processes of memory storage, recall, forecast and the benchmarks used to assess their performance.

3.1 One dimensional Grid-Based Spatial Representation

One dimensional grid cells serve as the foundational structure in our implementation of the Vector-HaSH model.

1. **Grid Representation:** A single dimensional grid module is represented by a one-hot encoded vector of length λ. A Grid is formed by the concatenation of all the single dimensional grid modules.
2. **Modular Structure:** The structure of our complete grid is modular. That is, our complete grid consists of multiple grid modules of different sizes defined by an array of *lambdas*. Then, the total size for a grid module is:

$$\text{Module Size} = \lambda$$

and the size of the full grid is:

$$\text{Grid Size} = \sum_{\text{all } \lambda} \lambda$$

This modular approach aids in handling larger grids by dividing the grid into manageable parts, where each module is processed independently during operations such as denoising or updating. In addition, the combination of these grid modules allows for more spaces in the environment to be mapped by our grid, therefore allowing the representation of a larger range of x coordinates without overlap.

3. **Temporal Data Parsing with Trailing Window Representation**
The time series data is parsed sequentially and split into tensors of size w. The hyperparameter w represents the window size, where the trailing length of each element is $w - 1$. Each tensor is stored as a single data entry, capturing both the current input and its preceding elements in a temporally sequential manner. This approach ensures that each data entry incorporates its historical context within the window.

Example: Given the time series data $[1, 2, 3, 4, 0]$ and $w = 4$, the parsed tensors are as follows:

$$[1, 0, 0, 0],$$
$$[2, 1, 0, 0],$$
$$[3, 2, 1, 0],$$
$$[4, 3, 2, 1],$$
$$[0, 4, 3, 2].$$

Here, each row corresponds to a tensor of size w, where trailing elements represent prior inputs padded with zeros when historical data is insufficient.

Including a memory trail of previous inputs is essential for capturing temporal dependencies and sequential patterns in the data. It allows the model to retain context from earlier time steps, enabling it to identify trends, recognize temporal relationships, and produce smoother, more stable predictions. By embedding this historical information, the model ensures that each prediction builds on past inputs, maintaining coherence and improving accuracy, especially for tasks where future outcomes are heavily influenced by prior observations.

3.2 Memory Encoding & Storage

Memory storage in the Grid4Cast is the same as in the Vector-HaSH model and involves the following steps (Figure: 2):

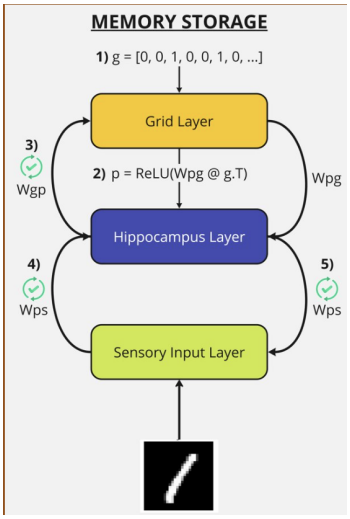


Figure 2: Storing process

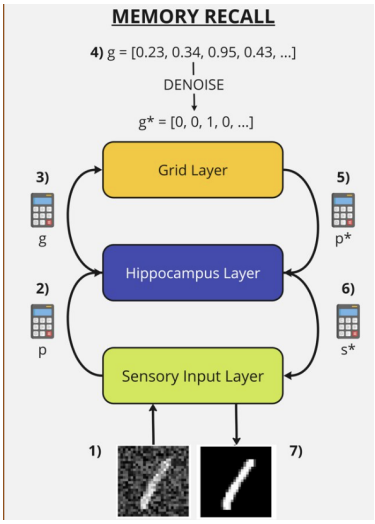


Figure 3: Recall process

Figure 4: Storing and recalling process in the Vector-Hash model

1. **Initialization of Weights:** The memory model initializes several weight matrices that connect the grid cells (g), hippocampal place cells (p), and sensory inputs (s). These connections form two crucial components of the model: the *scaffold* and *heteroassociation* mechanisms. The weights include W_{pg} (grid to place cells), W_{gp} (place to grid cells). These two weight matrices make up the 'memory scaffold', which is where the model stores it's memories. The other weight matrices are: W_{ps} (sensory to place cells), and W_{sp} (place to sensory cells). These matrices provide the hetero-association of the sensory input to the stored memory in the scaffold.

   W_{pg} is initialized using a normal distribution, with hyperparameters c (Section:A.2) and var determining the mean and variance, respectively. After initialization, W_{pg} is fixed and does not change during the learning process, while the other weights are initialized to 0 and are updated as memories are stored.

2. **Memory Storage:** First, given a coordinate, we determine the state of the grid vector g. With this, we can compute the input into the hippocampus layer, p, using W_{pg}:

$$p = \text{ReLU}(W_{pg} \cdot g)$$

Once we have both the grid vector g and hippocampus place cell activation p, we can begin constructing the scaffold by updating the weights:

$$W_{gp} \leftarrow W_{gp} + \frac{g \cdot p^T}{\|p\|^2 + \epsilon}$$

Note that we can use the sensory input s, along with the newly calculated p to update the weights that handle hetero-association of the sensory input to the location in the scaffold:

$$W_{sp} \leftarrow W_{sp} + \frac{s \cdot p^T}{\|p\|^2 + \epsilon}$$

$$W_{ps} \leftarrow W_{ps} + \frac{p \cdot s^T}{\|s\|^2 + \epsilon}$$

The update process follows a Hebbian-like learning rule, where the weight matrices W_{gp}, W_{sp} and W_{ps} are incrementally updated using the following expressions:
These updates ensure that the weights are adjusted based on the correlation between the grid cells, place cells, and sensory inputs, with normalization to maintain stability in the learning process.

3.3 Memory Retrieval & Recall

Memory recall is the process of retrieving stored memories from noisy or incomplete sensory inputs.

(a) **Noisy Input Processing:** The noisy sensory input s is processed through the trained model to, first, compute the place cell activation p using Wps inside the following formula:

$$\text{ReLU}\left(W_{ps} \cdot s^\top\right)$$

The place cell activation p is used to infer the corresponding grid state g by multiplying it with the weight matrix W_{gp}. Mathematically, this is represented as:

$$g = W_{gp} \cdot p^\top$$

where W_{gp} is the weight matrix that maps place cell activations to the grid state representation, and $p^\top$ is the transpose of the place cell activation vector.

(b) **Denoising and Modular Reconstruction:** The inferred grid state is denoised using the modular grid structure. The denoising process functions as a contrast enhancement mechanism that selectively emphasizes stronger signals by amplifying their relative dominance, while attenuating weaker signals to reduce noise and highlight key features in the grid representation. This mechanism involves splitting the noisy grid into modules, normalizing each module by its maximum activation, and applying a non-linear scaling to amplify dominant activations. The result is a reconstructed clean grid state where the most strongly activated cells are enhanced, and weaker activations are attenuated. This denoising process ensures that the grid state returns to a valid and stable representation within the grid space, effectively filtering out noise and improving the accuracy of the grid's spatial representation. The denoising process follows these steps:

$$g_{\text{max}} = \max(\text{noisy\_g})$$

$$g_{\text{stable}} = \frac{\text{noisy\_g}}{g_{\text{max}}}$$

$$g_{\text{denoised}} = g_{\text{stable}}^b$$

return g_{denoised}

(c) **Sensory Input Reconstruction:** The denoised grid state is used to compute the place cell activation using W_{pg} in this equation:

$$\text{ReLU}\left(W_{pg} \cdot g^{\top}\right)$$

, which is then mapped back to the sensory layer using the weight matrix W_{sp}. The output is passed through a non-linear activation function, Tanh to produce the final reconstructed sensory input:

$$s = \tanh\left(W_{sp} \cdot p^{\top}\right)$$

Where s is the reconstructed sensory input.

3.4 Forecast Prediction Process

We implemented four different variants on the prediction work frame, with two different prediction scaling and two prediction variants:

(a) **Scaling with the input standard deviation and mean, predicting elements one by one, using the previous output as input.**
We start by normalizing the temporal data and then storing it in our Grid4Cast model using the method described 3.2. Then, given the last element of the stored time series, we predict the next elements in the forecast horizon (Fh). We normalize the input and store it's mean and standard deviation. To normalize a tensor, we compute its mean and standard deviation, then apply the normalization formula. The steps are as follows:

$$\begin{aligned} \text{mean} &= \text{torch.mean(tensor).item()} \\ \text{std} &= \text{torch.std(tensor).item()} \\ \text{norm\_tensor} &= \frac{\text{tensor} - \text{mean}}{\text{std} + \epsilon} \end{aligned}$$

where:

- mean is the average value of the tensor,
- std is its standard deviation,
- $\epsilon = 10^{-7}$ is a small stabilization factor to prevent division by zero, and
- norm_tensor is the resulting normalized tensor.

Then, we use this input to recall the most similar memory we have encoded in our model using the method described here:3.3. From this recalled element, we can get it's grid position. We shift the grid position by one and retrieve the memory in this next grid. We normalize this new prediction using the standard deviation and mean of the input, but accounting for the change in these values given the new predicted element.

Shifted Values Computation

To update the mean and standard deviation when a new element is added and an old one is removed, we used the following approach:
To update the mean directly:

$$\Delta\mu = \frac{x_{\text{new}} - x_{\text{old}}}{N}$$
$$\mu_{\text{new}} = \mu_{\text{old}} + \Delta\mu$$

To update the variance:

$$\Delta\sigma^2 = \frac{x_{\text{new}}^2 - x_{\text{old}}^2}{N} - (\mu_{\text{new}}^2 - \mu_{\text{old}}^2)$$
$$\sigma_{\text{new}}^2 = \sigma_{\text{old}}^2 + \Delta\sigma^2$$
$$\sigma_{\text{new}} = \sqrt{\sigma_{\text{new}}^2}$$

Where:
- N: The number of elements in the data tensor.
- x_{new}: The new value added to the data.
- x_{old}: The old value removed from the data.

- $\mu_{\text{old}}, \mu_{\text{new}}$: The mean before and after the update.
- $\sigma_{\text{old}}, \sigma_{\text{new}}$: The standard deviation before and after the update.

When a new element is normalized, the original value cannot be recovered without knowing the current mean and standard deviation. Incremental updates allow us to compute the new mean and variance using only the normalized new element, the removed element, and the existing statistics.
Then, to get the final prediction:

1. We replace the last $[w-1]$ elements of the scaled predicted tensor with the first $[w-1]$ elements of the the original input tensor.
2. Use this scaled prediction as the next input and repeat the process for the desired forecast horizon.

(b) **Scaling with the best linear transformation between input and prediction, predicting elements one by one, using the previous output as input.**
We start by normalizing the temporal data and then storing it in our Grid4Cast model using the method described 3.2. Then, given the last element of the stored time series, we predict the next elements in the forecast horizon (Fh). We normalize the input. Then, we use this noisy input to recall the most similar memory we have encoded in our model using the method described here:3.3. From this recalled element, we can get it's grid position and shift the grid position by one to retrieve the memory in the next grid. We then use linear regression to find the best scaling factor between the original input's first [w-1] elements and the normalized prediction's last [w-1 elements] with these steps:
1. **Target Values (y_{target}):** - Extract the first $[w-1]$ elements of the original input tensor (un-normalized):

$$y_{\text{target}} = x_{\text{original}}[:w-1]$$

2. **Input Features (X):** - Use the last $[w-1]$ elements of the normalized prediction:

$$X = x_{\text{t+1}}[1:w]$$

3. **Prepare the Augmented Matrix for Linear Regression**: - Prepare prediction trail (X_{aug}) for regression (note, indexes start at 1):

$$X_{\text{aug}} = \begin{bmatrix} x_{\text{t+1},2} & 1 \\ x_{\text{t+1},3} & 1 \\ \vdots & \vdots \\ x_{\text{t+1},w} & 1 \end{bmatrix}$$

- Add regularization (λ) and solve for weights (w) and bias (b):

$$\theta = (X_{\text{aug}}^T X_{\text{aug}} + \lambda I)^{-1} X_{\text{aug}}^T y_{\text{target}}$$

$$w = \theta[:-1], \quad b = \theta[-1]$$

4. **Scaling the Next Memory**: - Scale the recalled next memory using weights and bias:

$$x_{t+1scaled} = x_{t+1} \cdot w + b$$

To generate forecasts over a given horizon, we use each prediction as the input for the subsequent step, iteratively applying the same procedure for all required predictions.
We observed that towards the end of the forecast, the predicted values tended to plateau and stop updating. To address this issue, we introduced a slight Ridge regression regularization term to the linear regression model. The plateauing effect arises because, as the predictions progress, the values become increasingly similar. Since we compute the mean and standard deviation over the entire tensor, this results in identical updates and nearly identical tensors at each step. Specifically, the first predicted element is repeatedly added to the front of the tensor, further reinforcing the convergence to nearly identical values.
The inclusion of Ridge regression helps mitigate this behavior by penalizing overly similar coefficients, ensuring that the updates remain distinct and informative even as the forecast horizon extends.

(c) **Scaling with the input standard deviation and mean, predicting all sequential elements at once.**
In this implementation, we start by normalizing the temporal data and then storing it in our Grid4Cast model using the method described 3.2. Then, given the last element of the stored time series, we predict the next elements in the forecast horizon (Fh) by normalizing this particular element.
We use this noisy input to recall the most similar memory encoded in our model using the method described here:3.3. From this recalled element, we got it's grid position and shift the grid position by one Fh time to retrieve the memories sequentially stored all at once. Then, to scale the list of normalized predictions, we iterated through them one by one to scale them using the updated mean and std of the previous prediction as described here 2a

(d) **Scaling with the best linear transformation between input and prediction, predicting all sequential elements at once.**
In this implementation, we start by normalizing the temporal data and then storing it in our Grid4Cast model using the method described 3.2. Then, given the last element of the stored time series, we predict the next elements in the forecast horizon (Fh) by normalizing this particular element. Then, we use this noisy input to recall the most similar memory encoded in our model using the method described here:3.3. From this recalled element, we got it's grid position and shift the grid position by one Fh time to retrieve the memories sequentially stored all at once, which theoretically corresponds to the sequence of elements of the prediction in order. Then, to scale the list of normalized predictions, we iterated through them one by one to scale them using our linear regression method, finding the relationship between the previous scaled recall and the next one, as we did here: pred2b

4 Experiments

4.1 Image Datasets & M4 Datasets and M4 explanation

To assess the effectiveness and robustness of Grid4Case, we conduct extensive evaluations on the M4 datasets, a widely recognized benchmark for testing forecasting methods. These datasets consist of $100,000$ time series spanning diverse domains and exhibiting various temporal frequencies. These datasets are ideal for benchmarking as they reflect real-world challenges in time-series forecasting.

Frequency	Number of Series
Yearly	23,000
Quarterly	24,000
Monthly	48,000
Weekly	359
Daily	4,227
Hourly	414

Figure 5: Overview of the M4 dataset composition, showing the number of time series by frequency.

Performance on the M4 datasets was assessed using two standard metrics:

(a) Symmetric Mean Absolute Percentage Error (sMAPE): sMAPE measures how far off our forecast is from the actual value, as a percentage. It is 'symmetric' because it treats over-predictions and under-predictions equally.

(b) Mean Absolute Scaled Error (MASE): Evaluates how well our forecast performs compared to a naive baseline model.

4.2 Experiment Setup

Measured Smape and MASE across our 4 models, on many different temporal data series.

- **M4 evaluation benchmarks Performance measures: point forecast** The M4 competition used two key performance measures for point forecasts: the symmetric mean

absolute percentage error (sMAPE) and the mean absolute scaled error (MASE). sMAPE is scale-independent and intuitive, while MASE addresses some of sMAPE's limitations and has better mathematical properties.

(a) sMAPE

$$sMAPE = \frac{2}{h} \sum_{t=n+1}^{n+h} \frac{\mid Y_t - \widehat{Y}_t \mid}{\mid Y_t \mid + \mid \widehat{Y}_t \mid} * 100(\%)$$

(b) MASE

$$MASE = \frac{1}{h} \frac{\sum_{t=n+1}^{n+h} \mid Y_t - \widehat{Y}_t \mid}{\frac{1}{n-m} \sum_{t=m+1}^{n} \mid Y_t - Y_{t-m} \mid}$$

Y_t: value of time series at point t.
$\widehat{Y}_t$: estimated forecast
h: the forecasting horizon
n: the number of data points available in-sample
m: the time interval between successive observations considered by the organizers for each data frequency, i.e., 12 for monthly, 4 for quarterly, 24 for hourly and one for yearly, weekly and daily data.

Number of forecasts beyond the available data per time sequence type:

(a) Yearly: 6
(b) Monthly: 18
(c) Weekly: 13
(d) Daily: 14
(e) Hourly: 48

Minimum numbers of observations in the training test set:

(a) Yearly: 13
(b) Monthly: 42
(c) Weekly: 80
(d) Daily: 93
(e) Hourly: 700

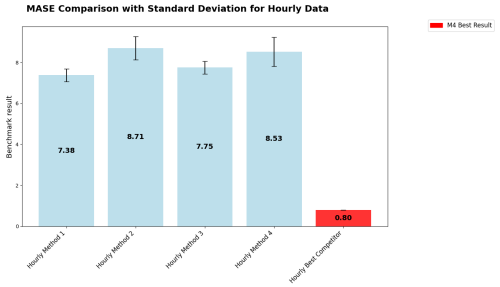
4.3 Results & Discussion

Results and Discussion

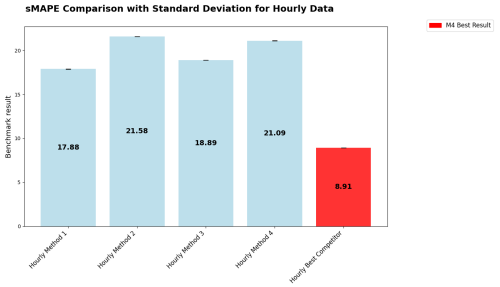
The performance of the four model variants was evaluated across ten trials and five temporal resolutions—hourly, daily, weekly, monthly, and yearly—using MASE (Mean Absolute Scaled Error) and sMAPE (Symmetric Mean Absolute Percentage Error) as evaluation metrics. The results are summarized as follows.

Hourly Data

For hourly data, the models exhibited relatively high errors, reflecting the challenges in capturing fine-grained temporal patterns in this case. The MASE values ranged between 7.38 and 8.71, with the lowest score achieved by Hourly Model 1. Similarly, sMAPE scores were high, ranging from 17.08% to 21.38%, with Hourly Model 4 producing the highest error. These results indicate that while the models were able to capture some patterns, noise and rapid changes in hourly data remain a challenge.



(a) MASE Comparison - Hourly

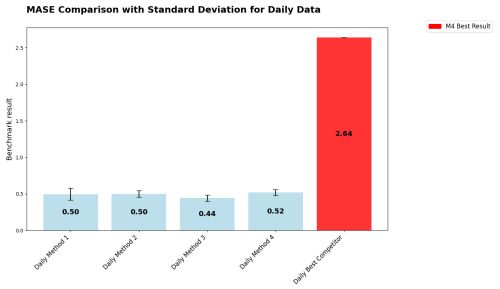


(b) sMAPE Comparison - Hourly

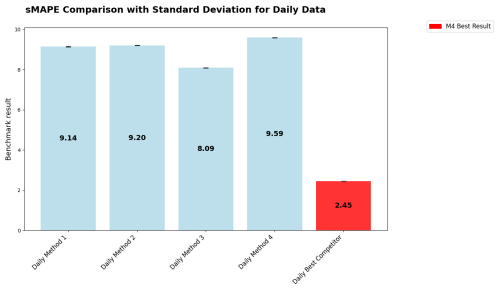
Figure 6: Performance Comparison for Hourly Data

Daily Data

In the case of daily data, the models demonstrated significantly improved performance. The MASE values were consistently low, ranging between 0.50 and 0.52, suggesting that all four model variants outperformed the naive baseline. The sMAPE scores ranged from 8.09% to 9.59%, indicating a strong predictive accuracy. The small standard deviations further highlight the stability of the models for daily forecasts.



(a) MASE Comparison - Daily

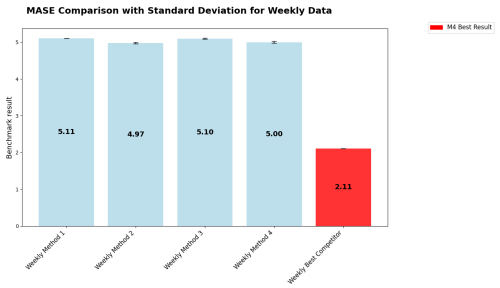


(b) sMAPE Comparison - Daily

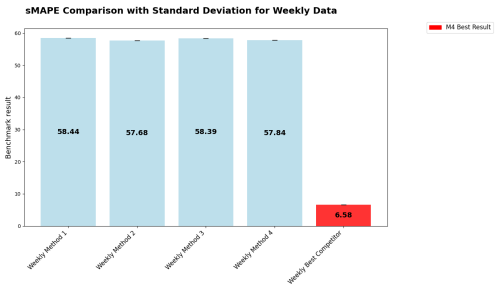
Figure 7: Performance Comparison for Daily Data

Weekly Data

The results for weekly data reveal a greater difficulty in predicting patterns at this frequency. The MASE values ranged from 4.97 to 5.11, showing limited variation between models. However, sMAPE scores were notably high, ranging between 57.68% and 58.44%, reflecting substantial forecast errors. These results suggest that weekly data introduces additional challenges, such as irregular trends or noise, which the models struggled to capture effectively.



(a) MASE Comparison - Weekly

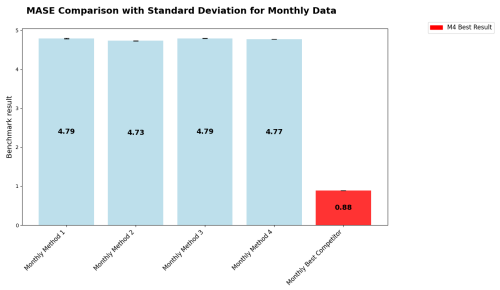


(b) sMAPE Comparison - Weekly

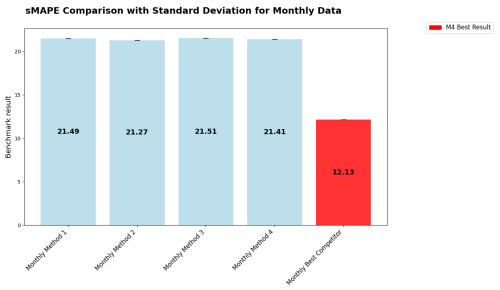
Figure 8: Performance Comparison for Weekly Data

Monthly Data

The monthly data results still indicate high errors but improved performance across the board compared to weekly data. The MASE scores range between 4.73 and 4.79, with minimal variation between the models. Similarly, the sMAPE scores fell within a narrow range, between 21.27% and 21.51%.



(a) MASE Comparison - Monthly

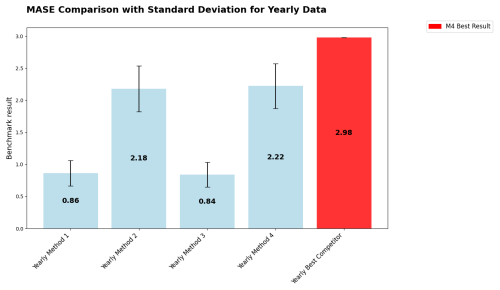


(b) sMAPE Comparison - Monthly

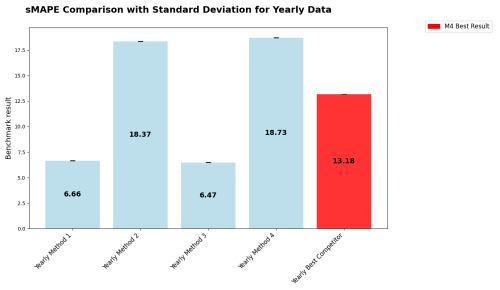
Figure 9: Performance Comparison for Monthly Data

Yearly Data

The yearly data results highlight strong performance relative to the naive baseline for the first and third models. The MASE scores ranged from 0.86 to 2.18, with the first and third models achieving the lowest scores. A similar pattern can be seen with the sMAPE values: the first and third models had comparable scores of 6.66 and 6.47, while the other two models suffered with 18.37 and 18.73. These results indicate that while the first and third models effectively captured long-term trends, there is still room for improvement in the second and fourth models. The similarity between the third and first models lies in their use of the same scaling method (adapted mean and standard deviation of the input). In contrast, the linear regression used for scaling in the second and fourth models may perform poorly here because it assumes a simple linear relationship, which is insufficient to capture the complex, non-linear long-term trends present in yearly data.

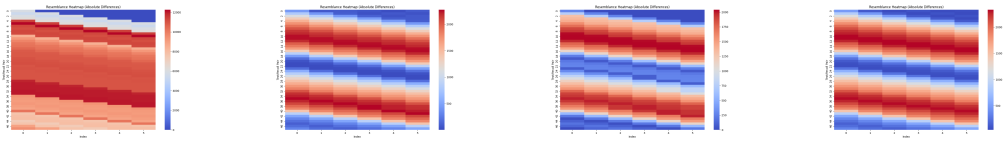


(a) MASE Comparison - Yearly



(b) sMAPE Comparison - Yearly

Figure 10: Performance Comparison for Yearly Data



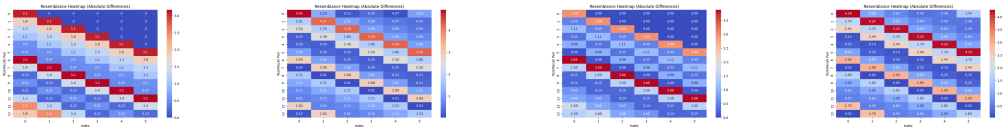
(a) Heat map for a sample of hourly data prediction for the first model variant.

(b) Heat map for a sample of hourly data prediction for the second model variant.

(c) Heat map for a sample of hourly data prediction for the third model variant.

(d) Heat map for a sample of hourly data prediction for the fourth model variant.

Figure 11: Heat maps demonstrating the correlation between predictions and actual values on the hourly temporal data, for the 4 model variant.



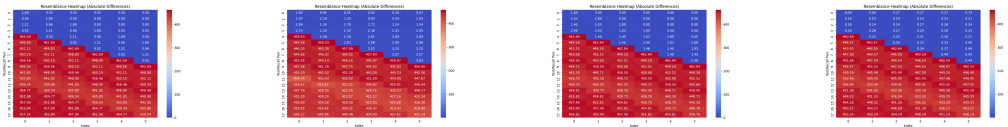
(a) Heat map for a sample of daily data prediction for the first model variant.

(b) Heat map for a sample of daily data prediction for the second model variant.

(c) Heat map for a sample of daily data prediction for the third model variant.

(d) Heat map for a sample of daily data prediction for the fourth model variant.

Figure 12: Heat maps demonstrating the correlation between predictions and actual values on the daily temporal data, for the 4 model variant.



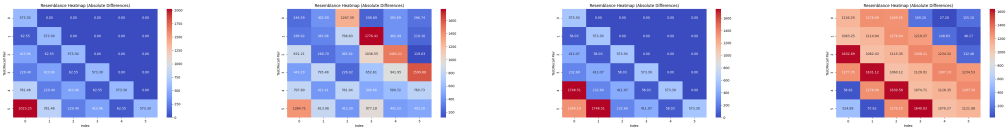
(a) Heat map for a sample of monthly data prediction for the first model variant.

(b) Heat map for a sample of monthly data prediction for the second model variant.

(c) Heat map for a sample of monthly data prediction for the third model variant.

(d) Heat map for a sample of monthly data prediction for the fourth model variant.

Figure 13: Heat maps demonstrating the correlation between predictions and actual values on the monthly temporal data, for the 4 model variant.



(a) Heat map for a sample of yearly data prediction for the first model variant.

(b) Heat map for a sample of yearly data prediction for the second model variant.

(c) Heat map for a sample of yearly data prediction for the third model variant.

(d) Heat map for a sample of yearly data prediction for the fourth model variant.

Figure 14: Heat maps demonstrating the correlation between predictions and actual values on the yearly temporal data, for the 4 model variant.

The heat maps illustrate the absolute differences in resemblance scores, where blue regions indicate strong similarity (low errors) and red regions highlight significant mismatches (high errors) between test/recall pairs and indices. These were calculated by computing the absolute differences between recall and test pairs across indices and visualizing the results in a matrix format. Rows correspond to test/recall pairs, while columns represent indices of comparison, enabling a clear depiction of error patterns. The similarity heat maps

highlight smoother and more stable patterns for annual data14, with fewer high-error regions and more uniform values (blue areas), reflecting improved consistency due to reduced recall density. In contrast, hourly heat maps show denser high-error regions (red areas) and periodic structures11, illustrating challenges from higher recall density, noise, and rapid changes in finer-grained data.

Additionally, these plots align with the evaluation metrics (MASE and sMAPE) discussed in the results. For hourly data, high-error regions correspond to relatively high MASE (7.38–8.71) and sMAPE (17.08%–21.38%) values, reflecting difficulties in capturing fine-grained temporal patterns. Daily data heat maps, with smoother patterns and fewer high-error regions, align with low MASE (0.50–0.52) and sMAPE (8.09%–9.59%), indicating strong predictive accuracy. Similarly, monthly heat maps, showing moderate stability and visible high-error regions, are consistent with moderate MASE (4.73–4.79) and sMAPE (21.27%–21.51%) scores. Lastly, yearly heat maps exhibit the most stable low-error regions, matching the strong performance of the first and third models with low MASE (0.86–2.18) and sMAPE (6.47%–6.66%). Together, the heat maps and metrics demonstrate how reduced recall density improves predictive stability across temporal resolutions for out models.

5 Conclusions

We investigated the performance of one-dimensional grid cells for temporal data predictions based on the Vector-HaSH model, introducing the Grid4Cast framework. Through experiments on the M4 benchmark datasets, we demonstrated Grid4Cast's ability to encode, recall, and predict temporal patterns, achieving strong performance on daily and yearly datasets with competitive sMAPE and MASE scores, while encountering challenges with hourly, weekly, and monthly data. The modular grid structure, with de-noising mechanisms, facilitated efficient memory storage and recall; however, the poorer performances with hourly, weekly and monthly datasets highlight areas for improvement.

5.1 Future Directions

For future directions, we should focus on understanding why the model achieves optimal performance at a small fraction of the theoretical capacity used, around 0.01. Investigating this behavior is crucial to uncover the role of grid sparsity, the effects of increasing grid size and stored memories, and potential saturation or interference in grid-to-place projections. This understanding could enable us to design a more compact and efficient model that maximizes performance while minimizing computational overhead, making it scalable for real-world applications.

Additionally, it would be interesting to explore why configurations using prime numbers perform better compared to those following Zipf's Law, despite the brain's reliance on Zipfian distributions in natural systems A.1. This deviation highlights potential unique structural properties of the dataset or model configuration, warranting deeper investigation. Testing the model on diverse temporal datasets, such as synthetic noise sequences, real-world noisy signals (e.g., stock prices, sensor data), and periodic patterns, would further evaluate its robustness and generalization.

To enhance prediction accuracy, we could explore fine-tuning strategies, including hyper-parameter optimization, adaptive learning rules, and alternative scaling methods beyond linear regression. Improving the model's compactness and performance is essential for efficiently handling complex, noisy temporal data, making it a strong candidate for practical, large-scale forecasting tasks.

6 Acknowledgments

We would like to express our deepest gratitude to Dr. Isabeau Prémont-Schwarz for his invaluable contributions to this work. His guidance was instrumental in the design of the algorithm, debugging, and the planning of experiments and tests. Dr. Prémont-Schwarz's unwavering support and mentorship provided us with not only technical expertise but also problem-solving approaches and practical methodologies that we believe will benefit us in the future. We are deeply appreciative of his ongoing dedication to our success.

References

[1] Sarthak Chandra, Sugandha Sharma, Rishdev Chaudhuri, and Ila Fiete. High-capacity flexible hippocampal associative and episodic memory enabled by prestructured "spatial" representations. *bioRxiv*, 2023.

[2] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. The m4 competition: 100,000 time series and 61 forecasting methods. *International Journal of Forecasting*, 36(1):54–74, 2020.

[3] Steven T. Piantadosi. Zipf's word frequency law in natural language: a critical review and future directions. *Psychonomic Bulletin Review*, 21(5):1112–1130, October 2014.

[4] Sugandha Sharma, Sarthak Chandra, and Ila R. Fiete. Content addressable memory without catastrophic forgetting by heteroassociation with a fixed scaffold. *arXiv*, 2022.

A Supplementary Material

A.1 Parameter Test for Np

We conducted a hyper-parameter search to find the ideal number of Np (place cells) for our model to project on using 3 different configurations. We measured the average cosine similarity and the standard deviation for each Np, for each configuration.15 For our model, we decided to use the first configuration, with Np=20.

- Configuration 1: For the first configuration, we set Lambda to be a list of the first 25 prime numbers, Nh = 1000 and Ng= 1060. The best value of Np for this configuration was around 15-20.
- Configuration 2: For the second configuration, we set Lambda to be a list of the first 70 prime numbers, Nh = 10000 and Ng= 10887. The best value of Np for this configuration was around 20.
- Configuration 3: For the third configuration, we set Lambda to be a list of numbers following Zipf's law, Nh = 1000 and Ng= 1077. The best value of Np for this configuration was around 43. The results for this configuration were the worst, which is intriguing because Zipf's Law suggests that in many natural systems, such as word frequencies in a language, the frequency of an item is inversely proportional to its rank. In other words, the most frequent item typically occurs about twice as often as the second most frequent item, three times as often as the third, and so on. The deviation observed in this case indicates that the distribution may not follow the expected pattern, potentially pointing to unique structural characteristics within the dataset or configuration. This unexpected behavior highlights the need for further analysis to uncover the factors influencing this outcome.[3]

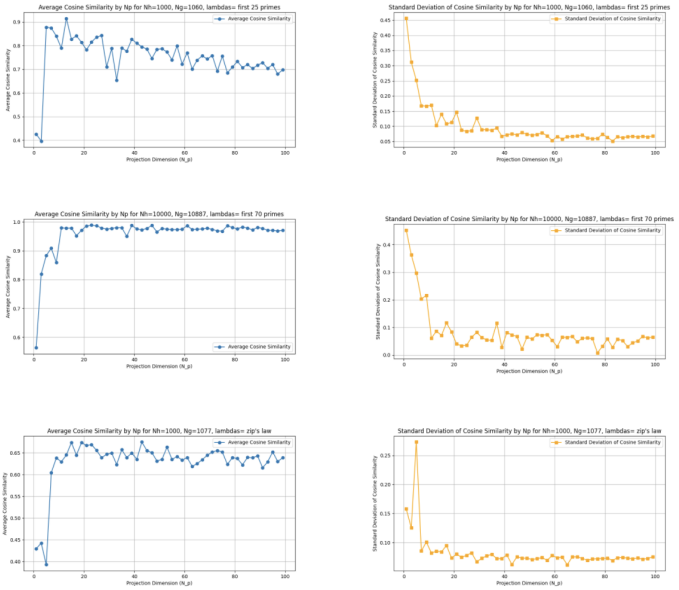


Figure 15: Results for the hyper-parameter search for the optimal number Np, for 3 different configurations.

A.2 Parameter c

To determine the hyper-parameter c for grid initialization, we used the projection dimension value (Np) and a variance set to 1.0.

$$c = -\sqrt{2 \cdot \text{var}} \cdot \text{erfinv}\left(\frac{1 - 2 \cdot \text{Np}}{N_h}\right)$$

A.3 With varying amounts of stored images:

We first wanted to analyze the relationship between the theoretical capacity of the grids used and the quality of our recalling process. To do so, we tested a different number of MNIST images in the grid with different numbers of Np. We hypothesized that the theoretical capacity was maybe dependent on Np, and not just this formula:

$$\text{Theoretical Capacity} = \text{Number of floating-point numbers in } W_{pg} = \text{num\_grid\_cells} \times \text{num\_place\_cells}$$

$$\text{Used Capacity} = w \times \text{num\_memories}$$

There is a clear relationship between the number of images stored and the quality of recall. The optimal performance is around 0.01 and decreases as we store more elements. These results also indicate that the capacity does not in fact depend on Np.

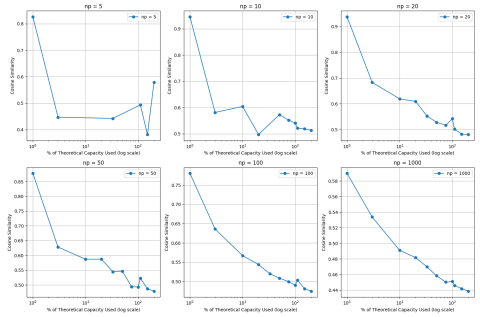


Figure 16: Tests for the theoretical capacity used (above 0.01).

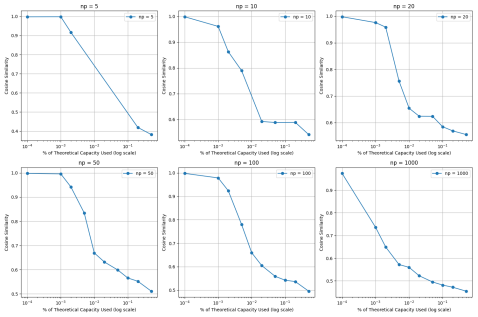


Figure 17: Tests for a smaller number theoretical capacity used.

Figure 18: Tests for the theoretical capacity used done on MNIST.