

Variants of GRNN for time-series predictions: dilated variant and hybrid model with CNN

Karel Šafr^{1,2*}, Patrik Hric¹ and David Hartman^{2,3}

¹*Department of Statistics and Probability, Faculty of Informatics and Statistics, Prague University of Economics and Business, nám. W. Churchilla 1938/4, Prague, 130 67, Czech Republic.

²Unicorn University, V Kapslovně 2767/2, Prague, 13000, Czech Republic.

³Institute of Computer Science of the Czech Academy of Sciences, Czech Academy of Sciences, Pod Vodárenskou věží 271/2, Prague 8, 18207, Czech Republic.

*Corresponding author(s). E-mail(s): karelsafr@gmail.com;
Contributing authors: patrikhric9@gmail.com; hartman@cs.cas.cz;

Abstract

General Regression Neural Networks (GRNN) are simple yet powerful nonparametric models for regression tasks. In this work, we investigate how GRNN can be adapted for time series forecasting by incorporating temporal decay into the similarity measure, as well as how its performance can be enhanced by combining it with convolutional encoders. We first introduce two novel time-decay GRNN variants that penalize distant observations either by modifying the distance or directly scaling the kernel. Second, we propose a new CNN→GRNN hybrid architecture that embeds lagged inputs through one-dimensional convolutional layers with pooling, bottleneck, and unit-norm normalization, followed by a GRNN operating in the learned embedding space. This architecture supports dilated convolutions, median-based initialization of the GRNN bandwidth, and efficient training with anchor subsampling and leave-one-out masking. We compare both proposed methods against baseline GRNN, linear regression, and shallow neural networks on both public market data (equity and crypto) and proprietary energy consumption and generation series. Across equities and commodity proxies, GRNN variants—especially the time-decay GRNN—achieved the lowest MSEs on most series, consistently outperforming linear and shallow neural baselines. On proprietary energy data, a compact ANN performed best,

while the proposed CNN-GRNN hybrid still surpassed classical baselines and added predictive value even with short training windows.

Keywords: neural networks, general regression neural network, time-decay models, GRNN, CNN, time-series predictions, hybrid model

1 Introduction

The general regression neural network (GRNN) is one of the approaches in neural networks based on the idea of general regression, as described in [Specht et al. \(1991\)](#). They are based on the principle of comparing sampled data of regressors, usually represented as vectors (in accordance with the literature on machine learning, we will refer to them as *features*), with existing data in the training dataset, again represented as a set of vectors, and the regression variable (*target variable*). The idea behind this predictor is to determine the distances between features, such as Euclidean or Manhattan distances. These distances are used in a kernel (a Gaussian kernel is one of the most commonly used) to weight the target using targets from the training dataset and weights computed from the distances between the observed and training features.

Technically, it is a supervised method and its main advantages are: I) it can be trained on small samples of data; II) it can handle nonlinearity (due to the Gaussian kernel); III) because they are parameter-free, there is no necessity for an optimisation algorithm like back-propagation, ADAM, etc. IV) It can handle noisy data. The disadvantages stem from the principles of their operation. These methods are computationally expensive, and there are few methods to improve their accuracy, unlike in classical neural networks (NN), where improvements can be achieved by adding more neurons, hidden layers to the NN, or modifying other parts of the NN's architecture.

The advantages mentioned above have led to the application of GRNN in a wide range of fields. In engineering, GRNN has been used for corrosion modelling of steel in soil [Ding et al. \(2019\)](#), predicting gearbox residual life [Zhang et al. \(2025\)](#), and estimating construction equipment maintenance costs [Yip and Fan \(2012\)](#). In materials science, it has been applied to predict the curing characteristics of carbon black-filled rubber blends [Kopal et al. \(2022\)](#), while in healthcare, it has been utilized for the early detection and prevention of oral cancer [Sharma and Om \(2015\)](#). GRNN has also been shown to be valuable in financial applications, including forecasting highly volatile financial time series [Dudek \(2015\)](#).

Time series forecasting stands out as one of the actively studied areas for GRNN. Its ability to capture nonlinear and nonstationary dynamics makes it attractive for forecasting tasks, especially when the data is noisy or limited in quantity. Numerous studies have proposed strategies to adapt GRNN to time-series problems. For example, authors of [Martínez et al. \(2022\)](#) discuss parameter tuning and preprocessing techniques to enable efficient time-series forecasts. Works such as [Martínez et al. \(2019\)](#) have demonstrated GRNN's competitive accuracy across a range of time series scenarios, while others such as [Dudek \(2015\)](#); [Martínez et al. \(2022\)](#) highlight its capacity to handle complex seasonal structures and multiple series simultaneously.

Despite these advantages, GRNN suffers from several significant limitations. The first limitation is its high computational cost. Unlike parametric models that compress knowledge into a small set of parameters, GRNN stores and relies on the entire training dataset to make predictions. This makes it computationally demanding, particularly when applied to large-scale or real-time forecasting tasks. Hybrid approaches have been proposed to address this issue, such as combining GRNN with decomposition techniques and metaheuristic optimization algorithms (e.g., in short-term load forecasting using empirical wavelet decomposition and sparrow search algorithms) [Fan et al. \(2023\)](#), or applying feature selection to reduce input dimensionality, as discussed in [Amato et al. \(2020\)](#).

The standard GRNN also has apparent limitations when applied to sequential data. First, the similarity measure treats all past observations equally, regardless of their recency, which is problematic in many forecasting settings where recent dynamics are more informative than distant history. While some variants indirectly address this issue by adjusting the bandwidth parameter or applying preprocessing steps, the model itself lacks a principled mechanism for incorporating temporal decay. Second, GRNN operates directly in the raw input space and therefore struggles to capture recurring motifs that may reappear at different time positions. Without the ability to learn shift-invariant or hierarchical features, the model often requires a large number of training samples to cover all possible variants of a pattern. To overcome these limitations, we propose two complementary extensions: a time-aware GRNN that explicitly integrates recency into the similarity function, and a hybrid CNN-GRNN model where convolutional filters first extract invariant features before kernel regression is performed in the learned embedding space.

Preliminary attempts to adapt GRNN to time series forecasting include approaches that reformulate the prediction task through lagged input-output pairs or adjust the bandwidth parameter to capture temporal dynamics better. For example, [Ahmed et al. \(2010\)](#) applies GRNN to time series by embedding the data into sliding windows of past observations, allowing the model to operate as a nonlinear autoregressor. [Martínez et al. \(2022\)](#) further extends this line of work by proposing strategies, such as rolling-origin bandwidth selection and data transformations, to handle trends and seasonality. While these methods illustrate how GRNN can be brought into a forecasting context, they differ from our approach. Rather than relying primarily on preprocessing or parameter tuning, we introduce explicit temporal decay into the similarity measure and combine GRNN with convolutional embeddings that provide shift-invariant representations.

Recent research has explored CNN-GRNN hybrid models across various application areas, although not specifically for time series forecasting. In contrast, our second proposed architecture is a hybrid CNN-GRNN architecture. Such architecture has already been used in a different context. In the study by [Hu et al. \(2019\)](#), the model was used to predict yarn quality attributes, including strength and irregularity. Here, CNN layers effectively extract spatial features from the input data, which GRNN subsequently processes for regression tasks. This approach demonstrated superior performance compared to traditional methods, such as multiple linear regression and backpropagation neural networks. Similarly, [Zhangla et al. \(2022\)](#) applied a

comparable CNN-GRNN hybrid architecture to estimate cavity volumes in manufacturing processes. While both studies demonstrate the strong predictive capabilities of CNN-GRNN models, they are based on static datasets without sequential or temporal components, which limits their direct applicability to time series forecasting problems.

In contrast to this approach, our second proposed architecture represents a hybrid CNN-GRNN architecture designed explicitly for sequential forecasting tasks. By placing a convolutional encoder in front of the GRNN, the model learns shift-invariant embeddings that capture recurring temporal motifs regardless of their position in the lag window. Pooling and optional bottlenecks further distil these features into compact representations that are robust to noise and reduce the risk of distance concentration. The GRNN then performs nonparametric regression in this learned embedding space, allowing it to interpolate between past trajectories more effectively. Together, these design choices yield a model that combines the expressiveness and efficiency of convolutional encoders with the sample-efficient, memory-based estimation of GRNN, resulting in a forecasting framework that is both flexible and data-efficient.

The remainder of the paper is organised as follows. Section 2 reviews the original GRNN formulation and its connection to kernel density estimation. Section 3 introduces our first proposed architecture based on time-decay extensions that incorporate temporal recency into the similarity measure. Section 4 presents the proposed CNN-GRNN hybrid architecture, which combines convolutional feature extraction with nonparametric regression in the embedding space. Section 5 describes the datasets, experimental design, and benchmark models used for evaluation, while Section 6 reports the empirical results. Finally, Section 7 concludes with a summary of findings and directions for future work.

2 General Regression Neural Network

The General Regression Neural Network (GRNN) was first introduced by Specht in 1991 [Specht et al. \(1991\)](#) as a kernel-based nonparametric regression method. It builds upon the Probabilistic Neural Network (PNN), introduced in [Specht \(1990\)](#), which was initially designed for classification, by extending kernel density estimation to continuous output prediction. Whereas PNN estimates the posterior probabilities of discrete classes, GRNN estimates the conditional expectation of a continuous response variable given the input.

Our work is based on the derivation of the GRNN model presented in [Specht et al. \(1991\)](#), which we used to understand the proposed modifications better. Consider the classical regression arrangement of random variables. Let \mathbf{X} (features) be a random vector with range \mathbb{R}^p and Y (target) be a random variable with range \mathbb{R} . Let us assume that we know the joint distribution function $f(\mathbf{X}, Y)$. Additionally, assume we have a set of instantiations \mathbf{x} of a random vector \mathbf{X} . The goal of regression is to estimate the conditional mean of the target variable given a set of instances of features, formally

$$E[Y|\mathbf{X} = \mathbf{x}] = \frac{\int_{-\infty}^{\infty} Y f(\mathbf{x}, Y) dY}{\int_{-\infty}^{\infty} f(\mathbf{x}, Y) dY}, \quad (1)$$

Since the joint density is often unknown, GRNN approximates it using kernel density estimation from a sample of observations $\mathbf{x} \in \mathbb{R}^{N \times p}$ of random vector \mathbf{X} and a sample of observations¹ $y \in \mathbb{R}^N$ of a random variable Y constituting together a training set $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$ ². The estimated joint density is

$$\hat{f}(\mathbf{x}, y) = \frac{1}{(2\pi)^{\frac{p+1}{2}} \sigma^{p+1}} \cdot \frac{1}{N} \sum_{i=1}^N \exp \left[-\frac{(\mathbf{x} - \mathbf{x}_i)^T (\mathbf{x} - \mathbf{x}_i)}{2\sigma^2} \right] \cdot \exp \left[-\frac{(y - y_i)^2}{2\sigma^2} \right]. \quad (2)$$

Here, N denotes the number of training samples, p represents the dimensionality of the input space, and $\sigma > 0$ is the smoothing or bandwidth parameter that controls the width of the Gaussian kernels. Substituting $\hat{f}(x, y)$ into the conditional expectation formula (1) leads to the estimate of the $E[Y|\mathbf{X} = \mathbf{x}]$ denoted for simplicity as $\hat{y}(\mathbf{x})$.

$$\hat{y}(\mathbf{x}) = \frac{\frac{1}{N} \sum_{i=1}^N \exp \left[-\frac{(\mathbf{x} - \mathbf{x}_i)^T (\mathbf{x} - \mathbf{x}_i)}{2\sigma^2} \right] \int_{-\infty}^{\infty} y \exp \left[-\frac{(y - y_i)^2}{2\sigma^2} \right] dy}{\frac{1}{N} \sum_{i=1}^N \exp \left[-\frac{(\mathbf{x} - \mathbf{x}_i)^T (\mathbf{x} - \mathbf{x}_i)}{2\sigma^2} \right] \int_{-\infty}^{\infty} \exp \left[-\frac{(y - y_i)^2}{2\sigma^2} \right] dy} \quad (3)$$

Solving the integrals analytically (due to Gaussian kernels) yields the GRNN prediction:

$$\hat{y}(\mathbf{x}) = \frac{\sum_{i=1}^N \exp \left[-\frac{(\mathbf{x} - \mathbf{x}_i)^T (\mathbf{x} - \mathbf{x}_i)}{2\sigma^2} \right] y_i}{\sum_{i=1}^N \exp \left[-\frac{(\mathbf{x} - \mathbf{x}_i)^T (\mathbf{x} - \mathbf{x}_i)}{2\sigma^2} \right]} \quad (4)$$

We denote the squared Euclidean distance between the query point \mathbf{x} and training point \mathbf{x}_i as

$$r_i^2 = \|\mathbf{x} - \mathbf{x}_i\|_2^2 = (\mathbf{x} - \mathbf{x}_i)^T (\mathbf{x} - \mathbf{x}_i) \quad \text{for all } i = 1, \dots, N \quad (5)$$

and simplify the expression from the equation (4) as

$$\hat{y}(\mathbf{x}) = \frac{\sum_{i=1}^N w_i y_i}{\sum_{i=1}^N w_i}, \quad (6)$$

where the weights are Gaussian kernel functions of distance

$$w_i = \exp \left(-\frac{r_i^2}{2\sigma^2} \right), \quad i = 1, \dots, N. \quad (7)$$

The smoothing parameter σ plays a key role in balancing bias and variance. Smaller σ values concentrate weight on nearby points, considering predominantly more similar cases but risking overfitting. Larger σ values provide smoother, more stable

¹Note that although this is a vector from the perspective of samples, we do not use the bold notation commonly used for vectors to distinguish this sample from a sample of observations of random vector \mathbf{x} .

²Note that the index $i = 1, \dots, N$ on the vector \mathbf{x} iterates through the samples of \mathbf{X} rather than the components of random vector \mathbf{X} .

predictions, while averaging considers all targets from a training set with nearly identical weights. This formulation treats GRNN as a memory-based estimator, where no explicit training beyond storing samples and selecting σ is required. Predictions are performed by weighted averaging of training outputs, with weights reflecting the proximity of inputs in the input space.

The connection to the Probabilistic Neural Network (PNN) lies in the shared kernel density estimation framework. While PNN uses similar Gaussian kernels to estimate posterior probabilities for classification tasks, GRNN extends this idea to estimate continuous conditional expectations, enabling regression.

3 Time decay extensions of GRNN

The simplest way to make a GRNN time-aware is to modify the input specification, as is common in classical statistical models. If we choose a simple sliding window approach and fix its size to 3, then the example of utilising the GRNN for time-series forecasting is as follows. We take the time series z_1, \dots, z_n (the higher index is the newer time) and create a training set by creating pairs $(\mathbf{x}_i = (z_i, z_{i+1}, z_{i+2}), y_i = z_{i+3})$ for all $i = 1, \dots, n-4$. When subsequently estimating the element z_n or equivalently y_{n-3} from the vector \mathbf{x}_{n-3} , the GRNN calculates the distances w_i between \mathbf{x}_i and \mathbf{x}_{n-3} for all $i = 1, \dots, n-4$. The resulting estimate z_n is then calculated as the weighted average of the values y_i using the weights w_i for all $i = 1, \dots, n-4$. Different applications of GRNN for time series prediction subsequently differ in the choice of validation and the parameter σ , as seen in studies [Ahmed et al. \(2010\)](#), which involve choosing a fixed value of σ and applying k -fold validation.

Authors of [Martínez et al. \(2022\)](#) describe a variant of this approach while suggesting how to use GRNN to forecast time series alternatively. They apply the following procedure: (i) detrend the targets additively or multiplicatively, (ii) embed seasonal lags, and (iii) tune the global smoothing parameter σ by rolling-origin validation, see [Tashman \(2000\)](#). The formula for the Euclidean distance itself remains unchanged.

In contrast to iterative inclusion of time information using the rolling-origin method, we inject time information directly into the similarity measure via various methods to include time into the computation of weights given by Equation 7. For the definition, consider that each training point x_i corresponds to time t_i and query point x corresponds to time t . Let us also define Δt_i as $\Delta t_i := t - t_i$. We have two variants to modify the similarity measure.

3.1 Distance weighted variant

In the first distance-weighted variant, we modify the distance between the training points and the query point, given by Equation (5), by including a monotonic function $g(\Delta t_i)$, leading to Equation (8), which takes into account the strength of the effect of the training point \mathbf{x}_i as a function of the temporal distance from the query point \mathbf{x} . The original weights computation given in Equation (7) remains unchanged. Altogether, this leads to the following set of equations.

$$(r_i^g)^2 = \|\mathbf{x} - \mathbf{x}_i\|_2^2 + \lambda g(\Delta t_i), \quad w_i^g = \exp\left(-\frac{(r_i^g)^2}{2\sigma^2}\right), \quad \hat{y}^g(\mathbf{x}) = \frac{\sum_i w_i^g y_i}{\sum_i w_i^g}. \quad (8)$$

The new elements are the scaling constant $\lambda > 0$, which governs the overall strength of this decay, and the function $g(\cdot)$ that converts the age of a pattern into its penalty. Examples of suitable functions include the linear function $g(\Delta t) = \Delta t$, which increases the distance as the age of the training sample increases. Alternatively, the exponential function $g(\Delta t) = e^{\Delta t/\tau}$ can be used to model a situation where distances should grow faster with respect to time. Note that the individual terms in the expression for $(r_i^g)^2$ in the Equation (8) may have different magnitudes or even orders of magnitude. The value of the constant λ is used, among other things, to compensate for this difference.

3.2 Kernel weighted variant

In the second kernel-weighted variant, we keep the distances given by equation (5) unchanged, but modify the calculation of the weights using the decay factor $h(\Delta t_i)$, resulting in Equation (9). This leads to a multiplicative modification of the distances as a function of time. The final weight of a pattern is first determined by spatial similarity and then rescaled according to its "age" of this pattern.

$$r_i^2 = \|\mathbf{x} - \mathbf{x}_i\|_2^2, \quad w_i^h = \exp\left(-\frac{r_i^2}{2\sigma^2}\right) h(\Delta t_i), \quad \hat{y}^h(\mathbf{x}) = \frac{\sum_i w_i^h y_i}{\sum_i w_i^h}. \quad (9)$$

Unlike the distance-weighted form described by Equation (8), where the time component is included in the exponent (shifting the effective squared distance), here the temporal influence acts as an independent multiplicative coefficient, which preserves the shape of the original kernel and makes it easy to plug in any decay function $h(\cdot)$. For example, we can again consider the negative exponential function $h(\Delta t) = \exp(-\Delta t/\tau)$. This time, however, such a function multiplicatively reduces the importance of observations based on their age. Here, it may also be beneficial to use the step function to model intervals of importance. Let us assume that we use the exp function as a multiplicative coefficient h , i.e. $h(\Delta t_i) = \exp(-\Delta t_i/\tau)$. The corresponding weights can then be rearranged as follows.

$$\begin{aligned} w_i^h &= \exp\left(-\frac{r_i^2}{2\sigma^2}\right) h(\Delta t_i) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}_i\|_2^2}{2\sigma^2}\right) \exp(-\Delta t_i/\tau) \\ &= \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}_i\|_2^2}{2\sigma^2} - \Delta t_i/\tau\right) \\ &= \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}_i\|_2^2}{2\sigma^2} - \frac{\lambda' \Delta t_i}{2\sigma^2}\right) \end{aligned}$$

where $\lambda' = \frac{2\sigma^2}{\tau}$. Now let us consider that the function g is equal to a linear function, i.e., $g = \Delta t$. Then we obtain the following relationship.

$$w_i^h = \exp\left(-\frac{(r_i^g)^2}{2\sigma^2}\right), \quad (10)$$

This indicates that the kernel-weighted variant, which utilises an exponential decay function, is equivalent to the distance-weighted variant with a linear penalty function. In fact, the equivalence between these two variants can be established more generally by appropriately redefining the functions g and h . However, that is not the subject of our work.

3.3 Properties of the time weighted GRNN

Let us recall the properties of the above-defined variants of GRNN. In the distance-weighted variant, defined by the Equation (8), a monotone function of recency $gaugmentsthesquaredradius(\Delta t_i)$, whereas in the kernel-weighted variant, described by the Equation (9), we keep the classical Gaussian radius but scale each weight by an additional decay factor $h(\Delta t_i)$. Both defined designs have two practical advantages that make them beneficial for practical purposes.

1. **Recency control:** the user can set the half-life τ (considering an appropriate decay function) or learn it jointly with σ instead of relying on an implicit preference induced by σ alone.
2. **Model parsimony:** We maintain a single GRNN for all horizons, thereby avoiding one model per future step, as in direct strategies such as those defined in Section 3.

The proposed approaches/ideas are not purely original. Similar ideas can also be observed in other models. e.g., the authors of the study [Du et al. \(2020\)](#) use space decay in geospatial data. Time-decay weighting has also been explored in several nonparametric models. Early work in EV-load forecasting introduced a time-weighted dot-product k-NN similarity, where recent observations receive exponentially larger weights. However, the temporal factor is multiplied only after the dot product and must be tuned ad hoc for each horizon. More recently, the prognostics literature has turned to time-weighted kernel density estimators that rescale KDE contributions by an exponentially decaying factor, again as an external multiplicative term ([Zhang et al. \(2025\)](#)).

4 Hybrid 1-D CNN-GRNN architecture

GRNNs operate directly in the input space and lack mechanisms for extracting shift-invariant or hierarchical representations, which can limit their ability to model complex or recurring temporal patterns. To address this, we first employ a CNN to transform the raw time-series data into an embedding space that captures salient, shift-invariant features. The GRNN then performs regression within this embedding space, enabling more accurate predictions by leveraging a richer and more structured representation of the temporal dynamics.

First, let's take a look at the structure of a convolutional layer. Let $\{(\mathbf{X}_i, y_i)\}_{i=1}^N$ be the training set, where $\mathbf{X}_i \in \mathbb{R}^{T \times C}$ is a lag window with T lags and C input channels ($C=1$ for a single univariate series) and $y_i \in \mathbb{R}$ is the target.

4.1 Convolutional encoder

The encoder is a stack of L one-dimensional convolutions with same padding, optional dilations, followed by a pooling operator, an optional linear bottleneck, and an ℓ_2 unit-normalization. Dilated convolutions exponentially increase the receptive field without inflating the number of parameters, a property that has proved crucial for modeling long-range temporal dependencies in speech and sequence modeling (Chen et al. (2014)). Denote $\mathbf{H}^{(0)} = \mathbf{X} \in \mathbb{R}^{T \times C}$ and for each layer $\ell = 1, \dots, L$ let m_ℓ be the kernel length, $r_\ell \in \mathbb{N}$ the dilation rate, K_ℓ the number of filters, and $\rho(\cdot) = \max(0, \cdot)$ the ReLU. The ℓ -th convolutional block computes

$$\mathbf{H}_{k,t}^{(\ell)} = \rho\left(b_k^{(\ell)} + \sum_{s=0}^{m_\ell-1} f_{k,s}^{(\ell)} \mathbf{H}_{t+r_\ell s}^{(\ell-1)}\right), \quad k = 1, \dots, K_\ell, \quad t = 1, \dots, T, \quad (11)$$

where $f_{k,s}^{(\ell)}$ denotes the kernel weight at position s of the k -th filter in layer ℓ , and $b_k^{(\ell)}$ is the corresponding bias term. Note also that for elements for which indices are out of range are handled using the padding function (“same”) ³. The convolutional layers act as pattern extractors: each filter responds to specific local motifs in the time series, and the resulting feature maps indicate both the type of pattern detected and its temporal location.

After the last convolutional layer, we apply a pooling operator P defined as follows.

$$\mathbf{u} = P(\mathbf{H}^{(L)}) = \begin{cases} \text{GAP: } u_k = \frac{1}{T} \sum_{t=1}^T \mathbf{H}_{k,t}^{(L)} \in \mathbb{R}^{K_L}, & (\text{GlobalAveragePooling1D}) \\ \text{FLAT: } \text{vec}(\mathbf{H}^{(L)}) \in \mathbb{R}^{K_L T}, & (\text{Flatten}) \end{cases} \quad (12)$$

where $\text{vec}(\cdot)$ denotes the vectorization operator, which reshapes the two-dimensional activation map $\mathbf{H}^{(L)}$ into a one-dimensional vector of length $K_L T$. This step serves as a summarizer into a more compact representation. Such a fixed-size embedding captures the most relevant global patterns, making it ideal for feeding into a regression head. Note that while pooling can normally destroy temporal information, in our case, we apply it locally or in a causal manner, ensuring that each time step still has its own embedding. This way, we preserve a sequence of embeddings that retain temporal ordering and summarize local context rather than the entire history. The GRNN then operates on this embedding sequence, modeling temporal dependencies in the transformed space.

The next step is an optional ⁴ linear bottleneck (no bias, no activation) defined as follows.

$$\tilde{\mathbf{z}} = \begin{cases} \mathbf{W} \mathbf{u} \in \mathbb{R}^q & \text{if a bottleneck dimension } q \text{ is specified} \\ \mathbf{u} & \text{otherwise} \end{cases} \quad (13)$$

³Our code: `Conv1D(padding="same", dilation_rate=r_l, activation="relu")`; all convolutional kernels and biases are *initialized to ones*, and an optional L_2 penalty may be applied.

⁴used if `q = bottleneck_dim` is not `None` in the implementation

where $W \in \mathbb{R}^{q \times d_u}$ is the trainable weight matrix of the bottleneck layer, mapping the pooled representation u of the dimension $d_u \in \{K_L, K_L T\}$ into a lower-dimensional embedding $\tilde{\mathbf{z}} \in \mathbb{R}^q$. No bias or activation is applied, so the bottleneck acts as a linear projection. The optional linear bottleneck projects the pooled features into a lower-dimensional subspace. This reduces computational cost in the subsequent GRNN stage and alleviates the risk of distance concentration in very high-dimensional spaces. By constraining the embedding dimensionality, the bottleneck also acts as a regularizer, encouraging the network to learn compact and discriminative representations that are well-suited for distance-based regression.

Finally, we apply ℓ_2 unit-normalization, projecting all embeddings onto the unit sphere. This step removes scale variability and ensures that distances are computed in a stable and comparable range across samples. In practice, normalization regularizes the geometry of the embedding space, making the GRNN kernel less sensitive to variations in magnitude and more focused on meaningful differences in feature direction.

$$\mathbf{z} = \frac{\tilde{\mathbf{z}}}{\|\tilde{\mathbf{z}}\|_2} \in \mathbb{R}^d, \quad d = \begin{cases} q, & \text{if bottleneck is used,} \\ K_L \text{ (GAP) or } K_L T \text{ (FLAT),} & \text{otherwise.} \end{cases} \quad (14)$$

We denote the resulting encoder by

$$\mathbf{z} = \varphi(\mathbf{X}; \theta) \quad (15)$$

where θ represents a set of parameters $\theta = \{f_{k,s}^{(\ell)}, b_k^{(\ell)}, \mathbf{W}\}$.

It is worth noting that our encoder implementation incorporates the following practical design choices, using the properties of the individual steps discussed above: (i) *GAP* is the default pooling; *Flatten* is an option. (ii) A low-dimensional bottleneck (default $q=32$) reduces the risk of distance concentration and improves memory/computation in the subsequent GRNN. (iii) Unit-norm makes the embedding lie on the ℓ_2 -sphere; hence the maximum pairwise Euclidean distance is 2, which we will exploit to bound σ .

4.2 GRNN in the embedding space with fixed anchors

For each training sample \mathbf{X}_i we use Equation (15) to compute its reference embedding $\mathbf{z}_i^{\text{ref}} = \varphi(\mathbf{X}_i; \theta)$ *once* and store it as a constant anchor. During training, these anchors are treated as constants: they are not updated when model parameters change, so gradients propagate only through the query encoder⁵. Given a query \mathbf{X} with embedding $\mathbf{z} = \varphi(\mathbf{X}; \theta)$, the GRNN uses Gaussian weights based on squared Euclidean distances in the embedding:

$$r_i^2 = \|\mathbf{z} - \mathbf{z}_i^{\text{ref}}\|_2^2, \quad w_i = \exp\left(-\frac{r_i^2}{2\sigma^2}\right), \quad \hat{y}(\mathbf{X}) = \frac{\sum_{i=1}^N w_i y_i}{\sum_{i=1}^N w_i + \varepsilon}, \quad (16)$$

⁵Our code: anchors are computed in `build()` and wrapped by `tf.stop_gradient`. This makes the memory fixed during training, while the query branch remains trainable.

with a tiny $\varepsilon > 0$ for numerical stability.

We apply leave-one-out masking to disable self-matches during training. If a query equals some anchor (same raw input window within tolerance τ_{self}), we set its self-weight to zero during training to avoid trivial memorization:

$$\text{if } \max_{t,c} |X_{t,c} - X_{t,c}^{(i)}| < \tau_{\text{self}} \Rightarrow w_i \leftarrow 0.$$

Reference subsampling is applied optionally during training to make kernel computations cheaper. To reduce the $\mathcal{O}(BN)$ cost per mini-batch (B queries, N anchors), we may randomly subsample $k \ll N$ anchors during training; inference uses all anchors.

4.3 Bandwidth σ : parameterization, bounds, initialization

The smoothing parameter σ is a trainable scalar internal parameter represented by an unconstrained raw variable $\alpha \in \mathbb{R}$ and mapped to a positive interval:

$$\sigma(\alpha) = \begin{cases} \sigma_{\min} + \text{softplus}(\alpha), & \text{if no upper bound,} \\ \sigma_{\min} + (\sigma_{\max} - \sigma_{\min}) \text{sigmoid}(\alpha), & \text{if } \sigma \in [\sigma_{\min}, \sigma_{\max}], \end{cases} \quad (17)$$

with a very small $\sigma_{\min} > 0$. When unit-norm is enabled, we set by default $\sigma_{\max} = 2.0$ since $\|\mathbf{z} - \mathbf{z}'\|_2 \leq 2$ for any unit vectors.

If the user does not provide initial values for σ (`init_sigma`), we compute embeddings for up to 4096 training windows with the current encoder, evaluate all pairwise distances, and set the initial σ to the median distance (snapshot of the geometry before training).

4.4 Training objective and optimization

Let $\hat{y}(\mathbf{X}_j)$ be the prediction for the j -th training pair provided by Equation (16) and y_j denotes the true target value corresponding to the input \mathbf{X}_j . We minimize the mean squared error:

$$\mathcal{L}(\theta, \alpha) = \frac{1}{N} \sum_{j=1}^N (\hat{y}(\mathbf{X}_j) - y_j)^2, \quad (18)$$

using Adam with early stopping and learning-rate reduction on plateau. By construction, gradients propagate through the query encoder $\varphi(\cdot; \theta)$ and through $\sigma(\alpha)$, while anchors remain fixed constants.

In the reference implementation we apply min-max scaling independently to inputs (per feature) and targets prior to training and inverse-transform predictions at inference time.

4.5 Dilations and receptive field

The encoder supports arbitrary dilation rates r_1, \dots, r_L . With stride 1 and same padding, the effective receptive field (measured in lags) of the L -layer stack equals

$$R_L = 1 + \sum_{\ell=1}^L (m_\ell - 1) r_\ell. \quad (19)$$

This value represents the number of past input lags that can influence a single output after L convolutional layers, i.e. it is the effective history length that the encoder can summarize for each step. For example, with two layers, $m_1 = m_2 = 3$ and $(r_1, r_2) = (1, 2)$ we obtain $R_2 = 1 + 2(1 + 2) = 7$ lags.

4.6 Summary of deviations added above vanilla CNN-GRNN

Compared to a naïve formulation that flattens all activations ($d=KT$) and learns σ by grid-search, the present implementation: (i) uses GAP or Flatten *before* an optional low-dimensional linear bottleneck; (ii) applies an ℓ_2 unit-normalisation; (iii) represents σ as a trainable scalar with principled bounds and a data-driven initialisation; (iv) keeps reference embeddings fixed (stop-gradient) and optionally performs leave-one-out masking and anchor subsampling during training.

5 Data

To evaluate the proposed time-decay variant of GRNN, along with the hybrid CNN-GRNN architecture, we utilise two complementary classes of time series. The first comprises public financial market data, which are noisy, volatile, and prone to structural changes, providing a challenging benchmark for predictive accuracy. The second consists of proprietary energy consumption and generation data, characterised by strong seasonalities and calendar effects, offering a contrasting setting where regular patterns dominate. Together, these datasets enable a broad assessment of the methods across scenarios with differing levels of noise, nonstationarity, and temporal structure. The subsections below provide a detailed description of the data sources, preprocessing, and feature construction.

5.1 Public market data

Daily and intraday series for selected liquid assets (equities and crypto), retrieved via `yfinance` (e.g., AAPL, BTC-USD). We use *adjusted close* (or *close*) and, where relevant, resample to the target frequency (daily or hourly). These series provide a volatile, non-stationary benchmark distinct from energy loads.

Equity and crypto price series were obtained directly from Yahoo Finance⁶ using the `yfinance` API⁷, which provides access to intraday and daily historical data with standardized fields (open, high, low, close, adjusted close, volume). All assets were

⁶ Accessible at <https://finance.yahoo.com/> [cited 2025-09-12]

⁷ Available at <https://github.com/ranaroussi/yfinance> [cited 2025-09-12]

downloaded with consistent time grids and aligned to a common local timezone to ensure comparability across series.

The selection of equities was not random but followed a deliberate economic logic. From each primary sector, we included one or more of the largest and most representative companies in terms of market capitalisation and global relevance. Technology is represented by Apple (AAPL), Microsoft (MSFT), Amazon (AMZN), and Alphabet (GOOGL); financial services by JPMorgan Chase (JPM) and Goldman Sachs (GS); healthcare by Johnson & Johnson (JNJ) and Pfizer (PFE); consumer and retail by Walmart (WMT); and automotive by Tesla (TSLA). In addition, we considered commodity proxies through gold (GLD) and crude oil (USO). This design ensures sectoral diversity and comparability, and avoids any bias that could arise from post hoc selection of assets based on predictive results.

5.2 Proprietary energy data

We consider anonymized quarter-hourly (15-minute) measurements of *consumption* (y_{spot}) and *generation* (y_{vyr}) aggregated over a fixed portfolio. The raw table contains a timestamp column \mathbf{ds} and the two targets $y_{\text{spot}}, y_{\text{vyr}}$. We enrich the data with calendar regressors:

- *hour of day* (0–23) and *day of week* (0–6) one-hot encoded,
- *weekend flag*,
- *lagged targets*: for the 15-minute frequency we add contiguous blocks of lags for the previous day and two days back. With 96 steps per day, we include lags $\{96, \dots, 119\}$ (24 quarter-hours, i.e., a 6-hour window one day back) and $\{192, \dots, 215\}$ (the analogous block two days back), for both y_{spot} and y_{vyr} . For hourly data, we use the analogous indices with 24 steps per day.

The private energy consumption and generation series were supplied by the company partner as part of our mutual research project TK05020142, see Acknowledgements section; although the data source differs, all preprocessing steps described below were applied identically.

5.3 Considered preprocessing

Including features. Raw time-stamped price data were enriched with a variety of engineered features before modeling. We first harmonized the sampling frequency (15-minute or daily) and filled missing observations by forward propagation. Calendar regressors were then added, including hour of day, day of week, weekend and holiday indicators, as well as their cyclic encodings using sine and cosine transforms. To capture short- and long-term dependencies, we introduced lagged versions of explanatory variables (daily lags) together with rolling statistics such as 24-hour moving means and standard deviations.

Cleaning and alignment. We drop rows with missing or infinite values after feature construction to avoid leakage via imputation across split boundaries. When training

on first differences (see below), the initial NA introduced by differencing is removed by aligning X and y accordingly.

Scaling. For baseline GRNN, see Section 5.5, we standardize predictors internally (z-score). For the CNN→GRNN hybrid, we apply min–max scaling independently to inputs and targets during training and invert the transform at inference.

5.4 Evaluation methodology

We evaluate *point forecasts* for both levels and (where relevant) first differences, at multiple sampling frequencies (quarter-hourly and hourly). Each experiment is anchored at a reference timestamp t_0 and uses a *single expanding hold-out* split:

- *Training window:* $[t_0 - 21 \text{ days}, t_0 - \Delta]$, where Δ is one sampling step (15 min or 1 h) to avoid look-ahead.
- *Test window:* $[t_0, t_0 + H]$, where H is either one full day (intraday day-ahead: 96 quarter-hour or 24 hourly steps) or a *multi-day horizon* (up to 30 days).

The same split is used across all models for a fair comparison. The entire workflow is repeated identically for quarter-hourly and hourly series.

To mitigate trend-driven extrapolation and stabilize local geometry, we train some models on *first differences*

$$\Delta y_t = y_t - y_{t-1},$$

and evaluate both on differences and on *reconstructed levels*. Let y_{anchor} be the last observed level in the training window (at $t_0 - \Delta$). Given predicted differences $\widehat{\Delta y}_t$ on the test window, levels are reconstructed by

$$\widehat{y}_t = y_{\text{anchor}} + \sum_{s=t_0}^t \widehat{\Delta y}_s.$$

We report errors primarily on levels; difference-based errors are provided as a diagnostic.

For each model and configuration, we compute MSE. Metrics are calculated on the full test window $[t_0, t_0 + H]$; when reporting multiple H (e.g., day-ahead vs. 30-day), we keep the split fixed and present both.

Finally, forecasts are generated in a *rolling-origin evaluation*: the forecast origin t_0 is advanced one day at a time, with the training window expanding accordingly. Hyperparameters are tuned within each training set, either via rolling-origin cross-validation or by setting aside the last portion of the training window. This procedure produces a sequence of strictly out-of-sample day-ahead forecasts that together form the evaluation sample and reflect realistic operational conditions.

5.5 Models under comparison

To place the proposed approaches in context, we benchmark them against a diverse set of forecasting models that represent common paradigms in the literature. These include classical linear methods, tree-based ensembles, shallow neural networks, as

well as the standard GRNN baseline. This comparison allows us to assess whether the time-decay GRNN variant and the CNN–GRNN hybrid offer tangible benefits over both traditional statistical models and more flexible machine learning techniques.

All models are trained on the same training window and evaluated on the same test window. For GRNN and LightGBM we sweep a small predefined grid and report performance per configuration; the best configuration (lowest MSE on the hold-out) is highlighted. For CNN→GRNN we vary filters, regularization strength, batch size, epochs, learning rate, and (where applicable) the upper bound on σ . We do not retune per time step; a single configuration produces the whole forecast path.

Linear regression (OLS). As a classical statistical baseline, we include ordinary least squares (OLS) regression applied to the same lagged and calendar features as the other models. OLS estimates the regression coefficients by minimizing the sum of squared errors, assuming a linear relationship between predictors and the target. While unable to capture nonlinear interactions, it provides a simple and interpretable benchmark that shows how much predictive accuracy can be achieved with purely linear dependence on past values and exogenous regressors.

GRNN (baseline GRNN). As a reference point, we include the standard GRNN with a Gaussian kernel, see Section 2, operating directly on lagged inputs and calendar features, serving as the baseline nonparametric model for time series forecasting. Inputs: tabular feature vector (calendar dummies + lagged targets). Training: on Δy_t with internal standardization of X . Prediction: differences $\widehat{\Delta y}_t$, then reconstruction to levels. Bandwidth σ : evaluated on a small grid (e.g., $\sigma \in \{0.025, 0.05, 0.3, 0.5, 1.5, 2, 2.5, 3, 3.5, 5, 8, 10\}$); we report metrics for each configuration and identify the best one.

GRNN extension with time-decay (Time-decay GRNN). Another model under comparison is the time-decay extension of GRNN introduced in Section 3. In this variant, the similarity measure is modified to incorporate temporal recency, either additively through a distance-weighted penalty or multiplicatively through a kernel-weighted decay. In both cases, the temporal half-life parameter and the kernel bandwidth σ are optimized jointly during training, so that the model learns not only the spatial notion of similarity but also how quickly past observations should lose influence. This provides a direct way of emphasizing recent patterns while still retaining the nonparametric interpolation principle of the original GRNN.

Hybrid 1-D CNN-GRNN architecture (hybrid CNN→GRNN). The second model is the proposed CNN→GRNN hybrid, where a convolutional encoder first maps lagged inputs into a learned embedding space and the GRNN then performs kernel regression using these embeddings as reference anchors, see Section 4. Encoder: a 1-D Conv stack with kernel size $m = 3$, filters in one or two layers (e.g., $(50, z)$ or (z) with $z \in \{5, 10, 15, 20, 30\}$), optional L_2 regularization and spatial dropout. We reshape $X \in \mathbb{R}^{N \times d}$ to $(N, d, 1)$, so convolution runs along a deterministic, grouped feature order (calendar features followed by contiguous blocks of lags). When we use

dilated filters in the encoder, dilation factors are treated as parameters of the same CNN→GRNN architecture. With stride 1 and same padding, the receptive field across L layers is

$$R_L = 1 + \sum_{\ell=1}^L (m_\ell - 1) r_\ell,$$

e.g., for two layers with $m_1 = m_2 = 3$ and $(r_1, r_2) = (1, 2)$ we obtain $R_2 = 7$ steps. GRNN head: Gaussian kernel in the learned embedding, with *trainable* scalar bandwidth σ bounded to $[\sigma_{\min}, \sigma_{\max}]$. σ is initialized from the median pairwise embedding distance and optimized jointly with the encoder by MSE. Training: Adam optimizer with early stopping and ReduceLROnPlateau; min-max scaling for X and y . Anchors (reference embeddings of the training set) are computed once and kept fixed (stop-gradient); an LOO mask prevents self-matches during training; optional sub-sampling of anchors reduces $O(BN)$ cost per batch. Outputs: direct level forecasts (no differencing in this variant).

A GRNN query scales as $O(N)$ in the number of anchors N . In CNN→GRNN, the encoder adds a small constant per batch, while the nonparametric head remains $O(BN)$ (batch size B). Optional anchor subsampling reduces training cost, while inference uses the full anchor set unless otherwise stated.

Linear and shallow neural baselines (ANN). As additional benchmarks, we include both linear and feed-forward neural models. The linear case is represented by ordinary least squares (OLS) regression applied to the same lagged and calendar features as other models. While simple, OLS provides a strong reference point for how much predictive structure can be captured by purely linear dependence on past values and calendar effects. To account for nonlinear relationships, we also consider shallow multilayer perceptrons (MLPs) with ReLU activations, optimized using mean squared error loss and monitored by mean absolute error. We experiment with both a single hidden layer (‘shallow’) and a slightly deeper variant, while keeping the architecture small enough to maintain comparability with other baselines. These models test whether modest nonlinear parameterizations can close the gap between linear regression and the more complex nonparametric approaches.

6 Results

In this section, we present the empirical evaluation of the proposed models alongside baseline methods. Forecasts were generated under a rolling-origin evaluation scheme across public financial series and proprietary energy series, with mean squared error (MSE) serving as the primary metric. The results are summarised in the following tables, grouped by domain. They are discussed with respect to the relative strengths of linear models, shallow neural networks, the standard GRNN, its time-decay extension, and the CNN→GRNN hybrid.

For all equity series, we used the same set of exogenous regressors capturing broad market conditions, namely the Nasdaq-100 ETF (QQQ), the volatility index proxy (VIXY), and the 10-year Treasury bond ETF (IEF). This ensures comparability across

assets and allows the models to incorporate systematic market effects. For the proprietary consumption and generation series, analogous regressors were constructed from weather data instead of financial indicators. Because the same regressors are applied consistently across all equities, any potential bias or distortion in these external variables affects all assets equally and therefore does not drive spurious differences in predictive performance between series.

We formally compare forecasting models by testing whether they achieve lower mean squared errors (MSE) than a set of baselines (OLS, ANN, GRNN). For each stock, we first compute squared residuals

$$e_{m,t}^2 = (\hat{y}_{m,t} - y_t)^2,$$

where y_t is the realized value and $\hat{y}_{m,t}$ the forecast of model m .

Pairwise statistical tests are then performed between every candidate model and each baseline. The test is a *paired one-sided t-test* applied to the difference of squared errors at each time point:

$$d_t = e_{m,t}^2 - e_{b,t}^2,$$

where m denotes the candidate model and b the baseline.

- Null hypothesis (H_0): $\mathbb{E}[d_t] \geq 0$. The candidate's average squared error is greater than or equal to the baseline's.
- Alternative hypothesis (H_1): $\mathbb{E}[d_t] < 0$. The candidate achieves a strictly lower average squared error than the baseline.

The test is paired because both models are evaluated on the exact same time points (identical test sets). Using the `ttest_rel` routine from `scipy.stats` with `alternative="less"`, we obtain p-values indicating whether the candidate significantly outperforms the baseline.

We report the resulting p-values in a matrix form: rows correspond to candidate models, and columns to baselines (OLS, ANN, GRNN). Small p-values provide evidence that the candidate model yields significantly lower forecasting error for the given stock.

The first results to show are for selected equities from the technology and financial services sectors. These assets were chosen as representative examples of highly liquid stocks with distinct volatility profiles. Table 1 reports the mean squared error (MSE) of all models on these series, providing an initial comparison of linear, neural, and GRNN-based approaches.

The next set of results covers the remaining equities, extending the analysis to consumer and retail (WMT), automotive (TSLA), healthcare (JNJ, PFE), and two commodity proxies (GLD and USO). These series complement the first group by adding assets with different dynamics: consumer demand cycles, high-volatility automotive stocks, defensive healthcare firms, and globally traded commodities. Table 2 reports the mean squared error (MSE) of all models on these series, highlighting performance across a broader spectrum of market behaviors.

The proprietary energy dataset contains quarter-hourly measurements of electricity consumption and production. Unlike equities and commodities, these series are

Table 1 Mean Squared Error (MSE) of all models across selected time series represented technology (APPL, MSFT, AMZN, GOOGL) and financial services (JPM, GS).

Model	APPL	MSFT	AMZN	GOOGL	JPM	GS
Linear regression (OLS)	6.67	3.85	6.00	3.37	11.20	5.01
ANN	4.92	4.89	5.08	4.04	9.31	3.23
GRNN baseline	3.22	1.96	2.85	2.34	5.48	1.71
Time-decay GRNN	3.21	6.00	2.05	1.81	8.35	1.60
hybrid CNN→GRNN	4.13	7.57	9.02	3.31	7.05	4.37

Table 2 Mean Squared Error (MSE) of all models across selected time series representing consumer and retail (WMT), automotive (TSLA), and commodities (GLD, USO).

Model	TSLA	WMT	JNJ	PFE	GLD	USO
Linear regression (OLS)	1.53	16.72	0.55	1.77	1.30	3.52
ANN	1.89	8.15	0.39	0.93	0.92	4.93
baseline GRNN	0.77	12.48	0.14	0.81	1.14	1.64
Time-decay GRNN	0.61	6.41	0.11	0.66	1.14	6.49
hybrid CNN→GRNN	1.01	8.82	0.32	1.26	1.05	6.44

characterized by strong seasonality and calendar effects rather than abrupt market-driven fluctuations. Table 3 summarizes the mean squared error (MSE) of all models on these targets, providing a contrast between financial forecasting tasks and energy load prediction.

The statistical tests (fig 1, 2, 3) reveal a consistent pattern across all stocks. Because the evaluation windows contain a large number of observations and the variability of squared errors is relatively small, even modest differences in forecasting accuracy are detected as statistically significant. Whenever a candidate model achieves a lower mean squared error than a baseline, the corresponding p-value is typically below 0.05 (and often below 0.01), providing strong evidence of superior predictive accuracy.

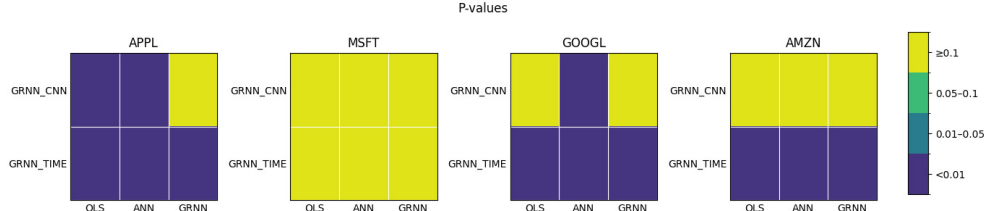


Fig. 1 Pairwise one-sided paired t-test (candidate vs. baseline) for APPL, MSFT, GOOGL and AMZN

Specifically, the GRNN_TIME model outperforms the classical baselines (OLS, ANN, GRNN) for the majority of stocks, with p-values indicating significance at the

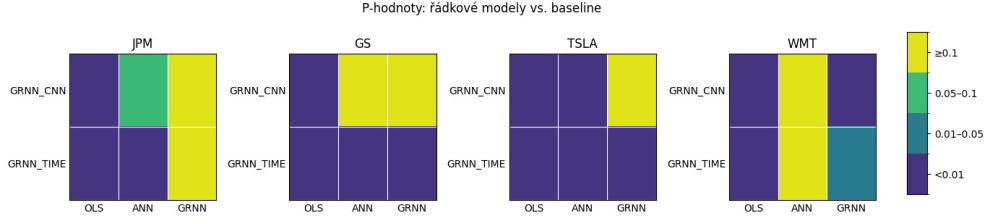


Fig. 2 Pairwise one-sided paired t-test (candidate vs. baseline) for JPM, GS, TSLA and WMT

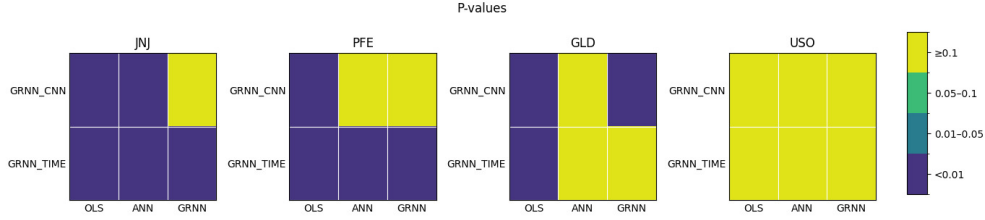


Fig. 3 Pairwise one-sided paired t-test (candidate vs. baseline) for JNJ, PFE, GLD and USO

1% level in almost all cases. The GRNN_CNN model also yields significant improvements in several instances, although the advantage is somewhat less systematic across all assets. For certain stocks (e.g., USO, MSFT, AMZN), differences are small and therefore not always significant, while in others (e.g., JNJ, GLD, JPM, TSLA) both advanced GRNN variants achieve clear and statistically significant gains relative to the baselines.

Overall, the results suggest that the proposed GRNN-based approaches consistently deliver forecasting errors that are at least as low as, and in many cases significantly lower than, those of standard econometric and neural network benchmarks.

Table 3 Mean Squared Error (MSE) of all models across selected time series representing electricity market descriptors (consumption and production)

Model	Consumption	Production
Linear regression (OLS)	3681.04	5247.37
ANN	3457.48	4517.03
baseline GRNN	4338.32	6087.83
Time-decay GRNN	4300.97	5679.36
hybrid CNN→GRNN	13362.65	17604.71

Across liquid equities and commodity proxies, members of the GRNN family delivered the best point forecasts in most series. The time-decay GRNN achieved the lowest MSE for AMZN, GS, TSLA, JNJ, and PFE, and was competitive for GOOGL

and WMT; The plain GRNN led on MSFT, USO, and JPM. For proprietary energy consumption and generation, a small feed-forward ANN performed best under our day-ahead rolling-origin design. The CNN→GRNN encoder, while not outperforming the strongest GRNN variants with only a 21-day training horizon, still consistently exceeded linear regression and shallow ANN baselines, indicating that convolutional embeddings add predictive value even in constrained settings. These findings suggest that explicit recency control is beneficial for fast-moving financial series, whereas smoother, quasi-seasonal energy loads favour simple parametric nonlinear baselines.

7 Conclusion

This paper investigated extensions of GRNN for time series forecasting. We considered two directions: (i) time-decay GRNN variants, where temporal recency is incorporated into the similarity measure, and (ii) CNN→GRNN hybrids, where convolutional encoders provide low-dimensional embeddings for the GRNN kernel regression. The CNN→GRNN framework supports pooling, bottlenecks, unit-norm normalisation, dilations, and a trainable kernel bandwidth initialised from median distances. All approaches were benchmarked against standard baselines including linear models, tree-based ensembles, and shallow neural networks, on both public financial series and private energy series at multiple sampling frequencies.

On public market data, both the plain GRNN and its time-decay extension consistently outperformed linear regression and shallow feed-forward networks, confirming the value of nonparametric similarity search for volatile financial series. The time-decay GRNN was particularly effective on assets such as AMZN, GS, TSLA, JNJ, and PFE, where recent information dominates predictive power. At the same time, the plain GRNN remained a robust default for MSFT, USO, and JPM. The CNN→GRNN encoder, although not surpassing the strongest GRNN variants under the short 21-day training horizon, still reliably outperformed linear and shallow ANN baselines, showing that convolutional embeddings contribute predictive signal even in constrained settings.

For the proprietary energy series, the best results were obtained by a compact feed-forward ANN, which aligns with their smoother, calendar- and weather-driven dynamics where simple nonlinear parametrisations suffice. Overall, the experiments demonstrate that all GRNN-based architectures and the CNN→GRNN hybrid clearly improve upon classical baselines, with time-decay extensions providing additional benefits for fast-moving financial assets. Additionally, convolutional embeddings offer a complementary path for further research on longer horizons or richer multivariate inputs.

The study demonstrates how GRNN can be systematically adapted to sequential data and how convolutional embeddings can offer a practical approach to integrating nonparametric memory-based estimation with modern deep learning techniques. The proposed CNN→GRNN hybrid offers a flexible yet data-efficient architecture for time series forecasting and provides a foundation for future extensions, such as integrating explicit time-decay mechanisms into the embedding model or applying the approach to multivariate forecasting tasks.

Supplementary information. There is no supplementary information attached.

8 Declarations

8.1 Conflicts of interest

The authors have no conflicts of interest to declare that are relevant to the content of this article.

8.2 Data availability statement

Data from the stock market is freely available from Yahoo Finance accessible at <https://finance.yahoo.com/> (cited 2025-09-12). The proprietary data on electricity consumption and production was provided by a cooperating company as part of project number TK05020142, subsidized by the Technology Agency of the Czech Republic for the purpose of testing models. However, it cannot be disclosed due to the company's licensing restrictions.

8.3 Ethics approval and consent to participate

Not applicable. This study does not involve human participants, animals, or sensitive personal data.

8.4 Acknowledgements

This work was supported by project n. TK05020142 operated with the state support of the Technology Agency of the Czech Republic within the THETA Programme. This study was partially supported by project "Research of Excellence on Digital Technologies and Wellbeing CZ.02.01.01/00/22_008/0004583" which is co-financed by the European Union.

References

- Ahmed, N.K., Atiya, A.F., Gayar, N.E., El-Shishiny, H.: An empirical comparison of machine learning models for time series forecasting. *Econometric reviews* **29**(5-6), 594–621 (2010) <https://doi.org/10.1080/07474938.2010.481556>
- Amato, F., Guignard, F., Jacquet, P., Kanevski, M.: On Feature Selection Using Anisotropic General Regression Neural Network (2020). <https://doi.org/10.48550/arXiv.2010.05744>
- Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., Yuille, A.L.: Semantic image segmentation with deep convolutional nets and fully connected crfs. *arXiv preprint arXiv:1412.7062* (2014) <https://doi.org/10.48550/arXiv.1412.7062>

- Ding, L., Rangaraju, P., Poursaei, A.: Application of generalized regression neural network method for corrosion modeling of steel embedded in soil. *Soils and Foundations* **59**(2), 474–483 (2019) <https://doi.org/10.1016/j.sandf.2018.12.016>
- Dudek, G.: Generalized regression neural network for forecasting time series with multiple seasonal cycles. In: *Intelligent Systems' 2014: Proceedings of the 7th IEEE International Conference Intelligent Systems IS'2014*, September 24–26, 2014, Warsaw, Poland, Volume 2: Tools, Architectures, Systems, Applications, pp. 839–846 (2015). https://doi.org/10.1007/978-3-319-11310-4_73 . Springer
- Du, Z., Wang, Z., Wu, S., Zhang, F., and, R.L.: Geographically neural network weighted regression for the accurate estimation of spatial non-stationarity. *International Journal of Geographical Information Science* **34**(7), 1353–1377 (2020) <https://doi.org/10.1080/13658816.2019.1707834>
- Fan, G.-F., Li, Y., Zhang, X.-Y., Yeh, Y.-H., Hong, W.-C.: Short-term load forecasting based on a generalized regression neural network optimized by an improved sparrow search algorithm using the empirical wavelet decomposition method. *Energy Science & Engineering* **11**(7), 2444–2468 (2023) <https://doi.org/10.1002/ese3.1465>
- Hu, Z., Zhao, Q., Wang, J.: The prediction model of worsted yarn quality based on CNN–GRNN neural network. *Neural Computing and Applications* **31**(9), 4551–4562 (2019) <https://doi.org/10.1007/s00521-018-3723-7>
- Kopal, I., Labaj, I., Vršková, J., Harničárová, M., Valíček, J., Ondrušová, D., Krmela, J., Palková, Z.: A generalized regression neural network model for predicting the curing characteristics of carbon black-filled rubber blends. *Polymers* **14**(4) (2022) <https://doi.org/10.3390/polym14040653>
- Martínez, F., Charte, F., Frías, M.P., Martínez-Rodríguez, A.M.: Strategies for time series forecasting with generalized regression neural networks. *Neurocomputing* **491**, 509–521 (2022) <https://doi.org/10.1016/j.neucom.2021.12.028>
- Martínez, F., Charte, F., Rivera, A.J., Frías, M.P.: Automatic time series forecasting with GRNN: A comparison with other models. In: *International Work-Conference on Artificial Neural Networks*, pp. 198–209 (2019). https://doi.org/10.1007/978-3-030-20521-8_17 . Springer
- Martínez, F., Frías, M.P., Pérez-Godoy, M.D., Rivera, A.J.: Time series Forecasting by Generalized Regression Neural Networks Trained With Multiple Series. *IEEE Access* **10**, 3275–3283 (2022) <https://doi.org/10.1109/ACCESS.2022.3140377>
- Specht, D.F., *et al.*: A General Regression Neural Network. *IEEE transactions on neural networks* **2**(6), 568–576 (1991) <https://doi.org/10.1109/72.97934>
- Sharma, N., Om, H.: Usage of Probabilistic and General Regression Neural Network for Early Detection and Prevention of Oral Cancer. *The Scientific World Journal*

- 2015(1), 234191 (2015) <https://doi.org/10.1155/2015/234191>
- Specht, D.F.: Probabilistic neural networks. *Neural networks* **3**(1), 109–118 (1990) [https://doi.org/10.1016/0893-6080\(90\)90049-Q](https://doi.org/10.1016/0893-6080(90)90049-Q)
- Tashman, L.J.: Out-of-sample tests of forecasting accuracy: an analysis and review. *International journal of forecasting* **16**(4), 437–450 (2000) [https://doi.org/10.1016/S0169-2070\(00\)00065-0](https://doi.org/10.1016/S0169-2070(00)00065-0)
- Yip, H., Fan, H.: A General Regression Neural Network model for construction equipment maintenance costs. In: 2012 7th International Conference on Computing and Convergence Technology (ICCCCT), pp. 1353–1357 (2012). <https://ieeexplore.ieee.org/abstract/document/6530551>
- Zhangla, X., Jiang, Y., Zhong, W., Zhang, H., Hou, J.: The prediction model of Irregular Cavity Volume based on CNN-GRNN neural network. In: 2022 2nd International Conference on Algorithms, High Performance Computing and Artificial Intelligence (AHPCAI), pp. 758–761 (2022). <https://doi.org/10.1109/AHPCAI57455.2022.10087396>
- Zhang, W., Zeng, J., Shi, H., et al.: Time-weighted Kernel Density for Gearbox Residual Life Prediction. *Scientific Reports* **15** (2025) <https://doi.org/10.1038/s41598-025-94924-z>