AI VIET NAM
@aivietnam.edu.vn

AI VIET NAM – AI COURSE 2025

LTSF-Linear: A simple, strong baseline for Long-Term Forecasts

Nguyễn Quốc Thái Hồ Quang Hiển
Nguyễn Phúc Thịnh Đinh Quang Vinh

I. Giới thiệu

Dự báo chuỗi thời gian, đặc biệt là trong lĩnh vực tài chính, là một trong những bài toán kinh điển và cũng đầy thách thức của ngành AI. Việc thành công trong dự đoán giá cổ phiếu không chỉ đem về cho ta lợi nhuận, mà còn là một công cụ cốt lõi để quản trị rủi ro. Tuy nhiên, thử thách của bài toán này thực sự không nằm ở việc đoán giá của ngày mai, mà là việc "nhìn xa" (Long-term Time Series Forecasting - LTSF). Dữ liệu tài chính nổi tiếng là cực kỳ "nhiễu" (noisy), phi tuyến tính (non-linear) và không có tính dừng (non-stationary). Dự đoán vài ngày tới đã khó, dự đoán hàng tuần hay hàng tháng là một bài toán ở cấp độ hoàn toàn khác.

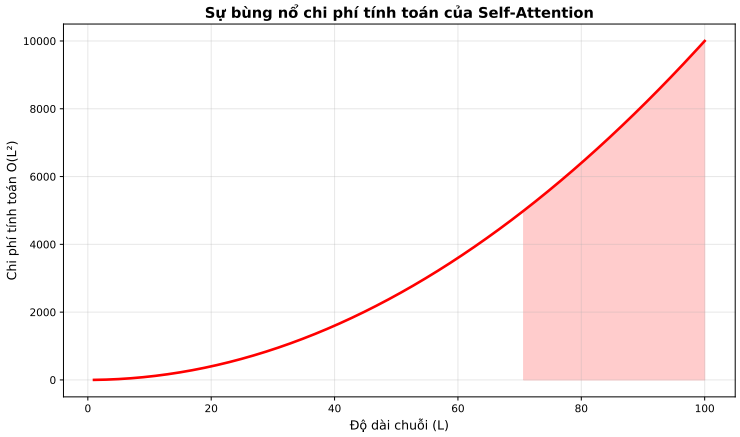
Hình 1: Sự biến động phức tạp và "nhiễu" của thị trường chứng khoán, một thách thức lớn cho các mô hình dự báo dài hạn.

Để giải quyết bài toán LTSF một cách hiệu quả, mô hình không chỉ đơn thuần là mở rộng cửa sổ dự báo. Thách thức cốt lõi nằm ở việc nắm bắt các phụ thuộc dài hạn (long-term dependencies). Trong dữ liệu tài chính, một sự kiện ở thời điểm t (ví dụ: giá hôm nay) có thể không chỉ phụ

thuộc vào $t-1$ (hôm qua), mà còn bị ảnh hưởng mạnh mẽ bởi các mô hình (patterns) đã xảy ra từ nhiều tháng, hoặc thậm chí nhiều năm trước. Ví dụ, tính mùa vụ (seasonality) có thể tạo ra các liên hệ cách nhau 12 tháng, hoặc các chu kỳ kinh tế (market cycles) có thể tạo ra các phụ thuộc kéo dài 3-5 năm. Các phương pháp dự báo truyền thống, thường thất bại trong việc "ghi nhớ" và liên kết các thông tin ở quá khứ xa xôi này. Do đó, yêu cầu cấp thiết là phải có một kiến trúc có khả năng "nhìn" và kết nối hiệu quả các điểm dữ liệu bất kể chúng cách xa nhau bao nhiêu, cho phép mô hình phân biệt đ

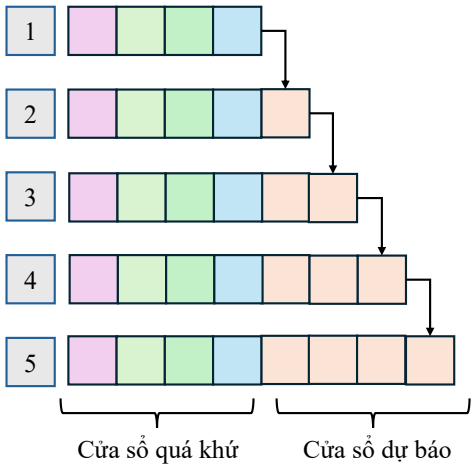
Khi nghe đến vấn đề phụ thuộc dài hạn, chúng ta thường nghĩ ngay đến lớp mô hình Transformer. Với cơ chế self-attention, các mô hình như Informer hay Autoformer hứa hẹn khả năng nhìn toàn bộ dòng lịch sử để tìm ra các mối quan hệ, phức tạp.

Nhưng có một vấn đề lớn với mô hình Transformer chính là độ phức tạp $O(L^2)$ của thuật toán. Khi L là độ dài chuỗi đầu vào (ví dụ $L = 720$, tương đương dữ liệu giao dịch 3 năm theo ngày), chi phí tính toán và bộ nhớ sẽ tăng vọt. Điều này khiến việc huấn luyện các mô hình Transformer cho chuỗi rất dài trở nên cực kỳ tốn kém, thậm chí là không khả thi trong thời gian thật.

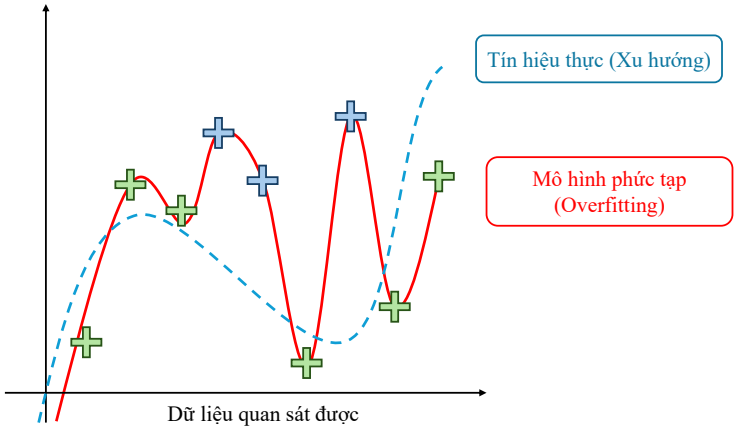


Hình 2: Sự bùng nổ về chi phí tính toán của cơ chế self-attention. Khi độ dài chuỗi L tăng lên, chi phí $O(L^2)$ khiến mô hình trở nên không thực tế cho các tác vụ dự báo rất dài. Vùng màu đỏ là khu vực mà VRAM sử dụng lớn hơn 8GB.

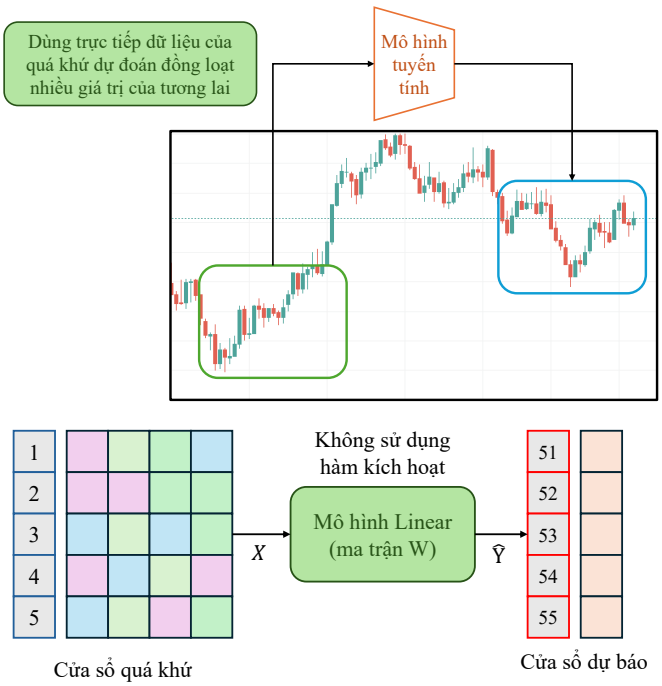
Điều này đặt ra một câu hỏi then chốt, đặc biệt là với dữ liệu nhiều nhiễu như tài chính: Liệu một kiến trúc siêu phức tạp có đang vô tình 'học vẹt' (overfit) chính cái nhiễu đó, thay vì nắm bắt tín hiệu thực sự?



Hình 3: Các mô hình chuỗi thời gian thường dự đoán tương lai bằng cách dự đoán tuần tự $t, t+1, \ldots, t+n$.



Hình 4: Minh họa hiện tượng Overfitting trong dữ liệu chuỗi thời gian. Các mô hình phức tạp thường 'học vẹt' một cách hoàn hảo các dao động ngẫu nhiên (nhiễu) có trong dữ liệu.



Hình 5: Kiến trúc của mô hình Linear. Thay vì dự đoán tuần tự các ngày trong tương lai, chúng chỉ cần chạy mô hình một lần.

Giữa bối cảnh đó, một nhóm mô hình 'quay về cơ bản' (back-to-basics) mang tên LTSF-Linear xuất hiện. Ý tưởng của chúng đơn giản đến bất ngờ: hãy mô hình hóa toàn bộ trục thời gian chỉ bằng một phép ánh xạ tuyến tính.

- Linear: Mô hình cơ sở, 'kéo' trực tiếp cửa sổ quá khứ sang tương lai bằng một ma trận trọng số duy nhất.
- DLinear: Cải tiến bằng cách phân rã (decompose) tín hiệu thành xu thế (trend) và dao động (seasonality), sau đó dùng hai mô hình Linear riêng biệt.
- NLinear: Bổ sung bước chuẩn hóa (normalization) để giảm lệch phân phối (distribution shift) trước khi đưa vào mô hình Linear.

Đặc biệt, trong các benchmark, những mô hình tuyến tính đơn giản này không chỉ nhẹ và ổn định, mà còn cho ra kết quả cạnh tranh, thậm chí vượt trội so với các Transformer phức tạp. Điều này dấy lên một giả thuyết mạnh mẽ: trong một miền 'nhiễu' như tài chính, việc tách lọc tín hiệu (như DLinear làm) có thể còn quan trọng hơn là cố gắng mô hình hóa mọi mối quan hệ phức tạp có thể có trong không gian bài toán.

Mục tiêu project: Project này không chỉ là một bài hướng dẫn để 'chạy lại' code mà còn là một project cơ sở để bạn hiểu việc xử lý dữ liệu chuỗi thời gian và đồng thời cải tiến chúng. Nhiệm vụ của bạn là:

- Hiểu rõ cách triển khai và so sánh Linear, NLinear, và DLinear trên dữ liệu tài chính thực tế.
- Phân tích (và phản biện) kết quả: Tại sao một mô hình đơn giản lại có thể hoạt động tốt? Giới hạn của nó ở đâu?
- Thử nghiệm và cải tiến: Bạn sẽ làm gì tiếp theo để cải thiện hiệu suất dự báo từ cái nền tảng này? (Ví dụ: thay đổi cửa sổ look-back, thử nghiệm với các nhóm cổ phiếu khác nhau, hay kết hợp các ý tưởng?)

Project này được thiết kế cho các bạn:

- Học viên AI/data science: Muốn áp dụng và 'thách thức' các mô hình time series trên dữ liệu tài chính phức tạp.

- Nhà phân tích định lượng (quants): Cần hiểu rõ các mô hình baseline mạnh, nhẹ để làm nền tảng cho các hệ thống phức tạp hơn.
- Bất kỳ ai tò mò: Giữa các lớp mô hình 'đơn giản' và 'phức tạp' trong machine learning, có nhất thiết càng phức tạp sẽ càng tốt hơn không?

Mục lục

Để việc đọc thuận tiện hơn, bảng sau liệt kê và chuẩn hoá các ký hiệu sẽ xuất hiện xuyên suốt nội dung bài.

Bảng Ký hiệu Toán học (LTSF)

Ký hiệu	Ý nghĩa
L	Độ dài cửa sổ quá khứ.
T	Độ dài dự báo tương lai.
B	Kích thước một batch.
$\mathbf{x} \in \mathbb{R}^L$	Vector input, chứa L giá trị quá khứ.
$\mathbf{y} \in \mathbb{R}^T$	Vector mục tiêu, chứa T giá trị thực tương lai.
$\hat{\mathbf{y}} \in \mathbb{R}^T$	Vector dự đoán, chứa T giá trị dự đoán tương lai.
x_t	Giá trị của chuỗi thời gian tại thời điểm t.
y_t	Giá trị thực tại bước tương lai thứ t.
$\hat{y}_t$	Giá trị dự đoán tại bước tương lai thứ t.
$W \in \mathbb{R}^{T \times L}$	Ma trận trọng số của mô hình Linear.
$b \in \mathbb{R}^T$	Vector bias của mô hình Linear.
ℓ	Mốc cục bộ (giá trị cuối x_t của $\mathbf{x}$) để tịnh tiến.
$\mathbf{1}_L, \mathbf{1}_T$	Vector chỉ chứa số 1, với độ dài L và T.
$\mathbf{x}'$	Vector input $\mathbf{x}$ đã được tịnh tiến (trừ ℓ).
$\hat{\mathbf{y}}'$	Vector dự đoán $\hat{\mathbf{y}}$ trước khi hoàn nguyên (cộng ℓ).
m	Kích thước cửa sổ của Moving Average.
$\mathrm{MA}_m(\cdot)$	Toán tử moving average với cửa sổ m.
x_t	Thành phần xu hướng của $\mathbf{x}$.
x_s	Thành phần mùa vụ của $\mathbf{x}$.
W_t, b_t	Ma trận trọng số và bias cho nhánh xu hướng.
W_s, b_s	Ma trận trọng số và bias cho nhánh mùa vụ.
$\hat{y}_t, \hat{y}_s$	Vector dự đoán cho nhánh Trend và Seasonal.

II. Kiến thức cơ bản về dự báo chuỗi thời gian

Trước khi đi vào mô hình, chúng ta đi qua cách đóng gói dữ liệu chuỗi thời gian để đưa vào mô hình học máy, kèm các khái niệm nền tảng về loại bài toán, đầu vào/đầu ra, và cách đánh giá các mô hình trên bài toán này.

II.1. Bài toán dự báo chuỗi thời gian là gì?

Một chuỗi thời gian đơn giản là một chuỗi các điểm dữ liệu được ghi lại theo thứ tự thời gian, ví dụ như giá cổ phiếu mỗi ngày, hoặc nhiệt độ mỗi giờ. Mục tiêu của bài toán dự báo là: dùng dữ liệu quá khứ để dự đoán dữ liệu tương lai. Chúng ta gọi độ dài của quá khứ mà mô hình nhìn vào là cửa sổ quá khứ (Look-back, L), và số bước tương lai cần dự đoán là chân trời dự đoán (Horizon, H).

Một cách tổng quát, chúng ta đang cố gắng tìm một hàm dự báo để học được quy luật này:

$$\underbrace{y_{t+1:t+H}}_{\text{đầu ra}} = \text{hàm dự báo}\Big(\underbrace{y_{t-L+1:t}}_{\text{quá khứ}}, \underbrace{x^{\text{đã quan sát}}_{t-L+1:t}}_{\text{biến phụ trợ đã quan sát}}, \underbrace{x^{\text{biết trước}}_{t+1:t+H}}_{\text{biến phụ trợ biết trước}}, \underbrace{s}_{\text{đặc trưng tĩnh}}\Big) + \underbrace{\varepsilon}_{\text{nhiễu}}.$$

Công thức này trông phức tạp, nhưng nó chỉ mô tả những gì chúng ta đã nói. Để dự đoán chính xác hơn (đầu ra màu xanh), chúng ta không chỉ dùng giá trị trong quá khứ (màu đỏ). Chúng ta cần cung cấp thêm thông tin, gọi là **biến phụ trợ** (covariates). Các loại này bao gồm **biến phụ trợ đã quan sát** (màu xanh lá), như lượng mưa hôm qua. Chúng cũng bao gồm các **biến phụ trợ biết trước** (màu tím), như ngày mai là chủ nhật. Và cuối cùng là các **đặc trưng tĩnh** (màu cam), như id của cửa hàng.

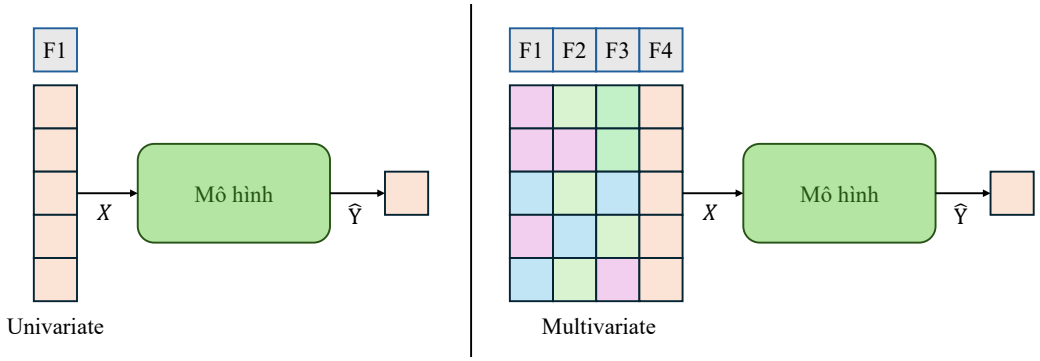
Khi nhìn vào một chuỗi thời gian, chúng ta thường có thể phân tách nó thành các thành phần. Các thành phần chính bao gồm **xu hướng (trend)**, là hướng đi chung của dữ liệu; **mùa vụ (seasonality)**, là các mẫu lặp lại có chu kỳ; và **nhiễu (noise)**, là các biến động ngẫu nhiên. Các mô hình khác nhau sẽ có cách xử lý các thành phần này khác nhau.

Hình 6: Phân tích chuỗi thời gian chúng ta sẽ sử dụng trong project này. Ta thấy rằng bài toán chúng ta có rất nhiều nhiễu, như ở plot cuối cùng có nhiều đường lên xuống.

II.2. Dữ liệu trông như thế nào?

Về cấu trúc, dữ liệu đầu vào có thể là đơn biến (univariate), nghĩa là chúng ta chỉ có một đặc trưng duy nhất (ví dụ: chỉ dùng chuỗi giá đóng cửa của cổ phiếu A). Hoặc, dữ liệu có thể là đa biến (multivariate), khi chúng ta sử dụng nhiều đặc trưng đầu vào cùng một lúc (ví dụ: dùng giá đóng cửa, khối lượng giao dịch của cổ phiếu A, và cả chỉ số VN-Index) để đưa ra dự đoán.

Một khái niệm liên quan là cấu trúc đầu ra (output). Chúng ta có thể dự đoán một chuỗi (ví dụ: chỉ dự đoán giá cổ phiếu A), hoặc dự đoán nhiều chuỗi cùng lúc (ví dụ: dự đoán giá của A, B, và C). Hình dưới minh họa sự khác biệt về cấu trúc đầu vào.

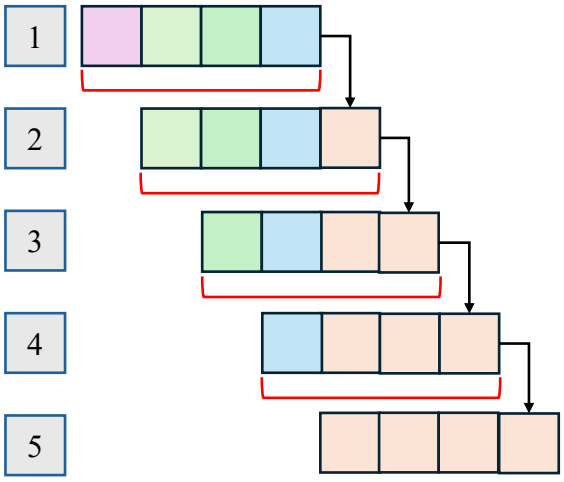


Hình 7: Hai cách biểu diễn đầu vào của bài toán chuỗi thời gian. Trong đó, cột cam là cột dữ liệu chứa các biến cần dự đoán.

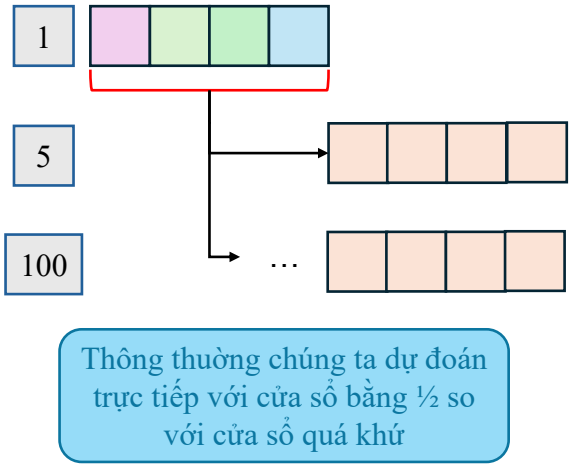
II.3. Chúng ta muốn dự đoán cái gì?

Khi nói về kiểu dự báo, chúng ta có thể dự đoán 1 bước (one-step) là giá trị ngay tiếp theo, hoặc **dự đoán nhiều bước (multi-step)** là một chuỗi giá trị trong tương lai ($H > 1$). Đây là mục tiêu chính của chúng ta ở trong bài toán này.

Chiến lược phổ biến thứ nhất cho bài toán multi-step là **đệ quy (recursive)**. Theo cách này, mô hình đầu tiên dự đoán bước thời gian $t+1$. Sau đó, kết quả dự đoán (giả) này được sử dụng làm đầu vào mới để mô hình tiếp tục dự đoán bước $t+2$. Quá trình này lặp đi lặp lại cho đến khi đủ H bước. Mặc dù trực quan, phương pháp này có một rủi ro rất lớn là tích lũy lỗi (compounding errors): một lỗi nhỏ ở bước $t+1$ có thể bị khuếch đại và làm sai lệch nghiêm trọng các dự đoán ở $t+H$.



Hình 8: Chiến lược dự đoán đệ quy: Mô hình dự đoán từng bước một, sử dụng kết quả dự đoán trước đó làm đầu vào cho bước tiếp theo.



Hình 9: Chiến lược dự đoán trực tiếp: Mô hình ánh xạ toàn bộ quá khứ đến toàn bộ tương lai trong một lần, tránh tích lũy lỗi.

Chiến lược thứ hai, hiệu quả hơn là **trực tiếp (direct)**. Với chiến lược này, mô hình được huấn luyện để học cách ánh xạ thẳng từ toàn bộ cửa sổ dữ liệu quá khứ (X) để dự đoán đồng thời *toàn bộ* H bước tương lai ($\hat{Y}$) chỉ trong một lần duy nhất. Phương pháp này hoàn toàn tránh được vấn đề tích lũy lỗi, vì mỗi dự đoán đều dựa trên dữ liệu thực tế. Đây chính là chiến lược cốt lõi được các mô hình Linear, NLinear, và DLinear sử dụng.

Cuối cùng, kết quả dự báo có thể là một **dự đoán điểm (point forecast)**, là một con số duy nhất như dự đoán ngày mai giá là 100. Hoặc, nó có thể là một **dự đoán xác suất (probabilistic forecast)**, trả về một khoảng giá trị, như dự đoán ngày mai giá nằm trong khoảng 95-105 với độ chắc chắn 90%.

II.4. Liên hệ tới các mô hình trong project này

Các mô hình trong project này đều áp dụng chiến lược dự báo trực tiếp. Mô hình **NLinear** tập trung vào việc chuẩn hóa (normalization) dữ liệu đầu vào để xử lý vấn đề khi dữ liệu trong tương lai trông khác nhiều so với quá khứ. Mô hình **DLinear** thì thực hiện theo một cách khác: nó tự động tách chuỗi thời gian thành hai phần là xu hướng và mùa vụ. Sau đó, nó dùng hai mô hình linear riêng biệt để học hai phần này, hoạt động rất tốt cho dữ liệu có xu hướng và mùa vụ rõ rệt.

III. Mô hình LTSF-Linear: Linear

Cơ chế: Linear mô hình hoá dự đoán dài hạn như một phép biến đổi tuyến tính dọc trục thời gian cho một biến (univariate). Với mỗi mẫu trong batch, ta có input $\mathbf{x} \in \mathbb{R}^L$ (cửa sổ quá khứ L bước) và output $\mathbf{y} \in \mathbb{R}^T$ (dự đoán T bước). Mô hình học ma trận $W \in \mathbb{R}^{T\times L}$ và bias $b \in \mathbb{R}^T$:

$$\hat{\mathbf{y}} \;=\; W\,\mathbf{x} + b \;\in\; \mathbb{R}^T.$$

Huấn luyện (DMS - Direct Multi-step). Tối ưu *đồng thời* cả T bước bằng MSE:

$$\mathcal{L}_{\text{MSE}} \;=\; \frac{1}{B\,T}\sum_{n=1}^{B}\sum_{t=1}^{T}\big(\hat{y}_t - y_t\big)^2.$$

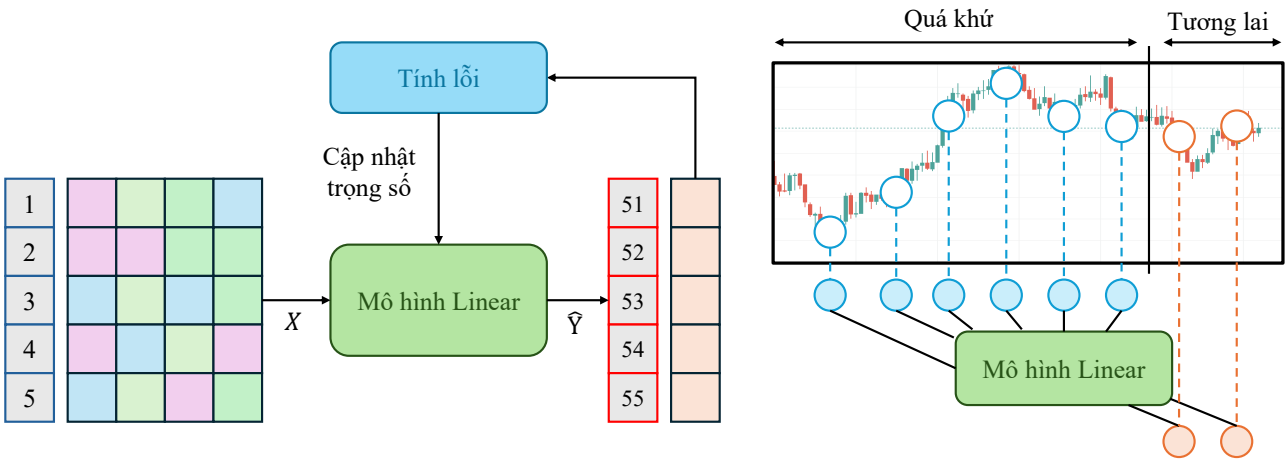
Trong đó:

- T: số bước dự đoán (horizon); B: kích thước batch.
- $\hat{y}_t$: giá trị **dự đoán** tại bước tương lai t; y_t: giá trị **thực**.
- MSE là **trung bình** sai số bình phương trên toàn bộ T bước và B mẫu.

Độ phức tạp. Tham số $\approx T \times L$ (cộng T bias); chi phí tính toán tỉ lệ $B \times T \times L \Rightarrow$ huấn luyện/suy luận nhanh, ổn định.

Lưu ý

Mô hình Linear trực tiếp biến đổi tuyến tính từ L bước gần nhất để dự đoán T bước kế tiếp, phù hợp khi chuỗi chúng ta đang xét ổn định. Nếu chuỗi có thuộc tính phi tuyến mạnh, có nhiều thay đổi đột ngột, hoặc cần mô hình hoá quan hệ đa biến, cân nhắc NLinear/DLinear hoặc các mô hình phi tuyến khác.



Hình 10: Quy trình chạy và huấn luyện của mô hình LTSF-Linear.

IV. Mô hình LTSF-Linear: NLinear

Cơ chế: Mô hình NLinear mở rộng lớp Linear cho dự đoán đơn biến bằng bước tịnh tiến dữ liệu về mốc cục bộ (**re-centering**) theo giá trị quan sát cuối của cửa sổ đang dự đoán, nhằm làm mô hình tập trung học sự biến thiên tương đối và giảm nhạy cảm trước dịch mức cộng tính (**additive level shift**). Hai thuật ngữ khó hiểu nói trên sẽ được diễn giải dưới đây.

Thuật ngữ:

- **Additive level shift**: toàn bộ các giá trị trong một đoạn thời gian cùng tăng hoặc cùng giảm một lượng như nhau. Nói cách khác, "mặt bằng" của chuỗi được đẩy lên hoặc hạ xuống nhưng hình dáng dao động (nhịp lên xuống, độ chênh giữa các điểm liền kề) hầu như không đổi.
- **Re-centering**: chọn một mốc tham chiếu gần nhất (thường là giá trị cuối cửa sổ), trừ mốc này khỏi mọi giá trị trong cửa sổ để đưa chuỗi về quanh mốc đó, huấn luyện mô hình trên phần độ lệch tương đối, rồi cộng lại mốc tham chiếu vào đầu ra. Cách này giúp mô hình ít bị ảnh hưởng khi mặt bằng dữ liệu dịch lên/xuống đồng đều.

Thiết lập dữ liệu. Với chuỗi $\{x_t\}$, tại thời điểm t tạo

$$\mathbf{x} = [x_{t-L+1}, \ldots, x_t] \in \mathbb{R}^L, \qquad \mathbf{y} = [x_{t+1}, \ldots, x_{t+T}] \in \mathbb{R}^T.$$

Trong đó (thiết lập):

- $\mathbf{x}$: input quá khứ dài L bước; $\mathbf{y}$: mục tiêu dự đoán T bước tương lai.
- $\mathbf{1}_L, \mathbf{1}_T$: vectơ với mọi phần tử đều bằng 1, kích thước L và T.

Công thức. Đặt mốc cục bộ $\ell = x_t$, thực hiện ba bước: *tịnh tiến* → *chiếu tuyến tính* → *hoàn*

nguyên:

$$\begin{aligned} \mathbf{x}' &= \mathbf{x} - \ell\,\mathbf{1}_L && \text{(tịnh tiến: đưa toàn bộ giá trị trong cửa sổ về quanh mốc cuối } \ell) \\ \hat{\mathbf{y}}' &= W\,\mathbf{x}' + b \\ \hat{\mathbf{y}} &= \hat{\mathbf{y}}' + \ell\,\mathbf{1}_T \;\in\; \mathbb{R}^T && \text{(hoàn nguyên: cộng lại mốc } \ell \text{ để trả dự đoán về mức gốc)} \end{aligned}$$

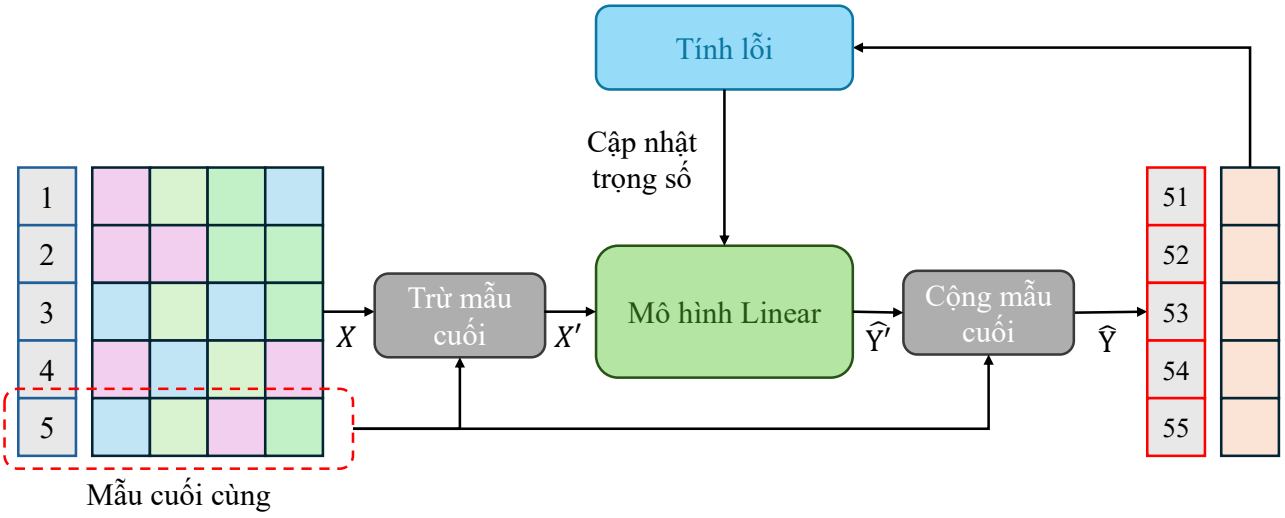
Huấn luyện (DMS). Tối ưu đồng thời cả T bước bằng MSE trên batch B:

$$\mathcal{L}_{\text{MSE}} = \frac{1}{B\,T} \sum_{n=1}^{B} \sum_{t=1}^{T} \left(\hat{y}_t - y_t\right)^2.$$

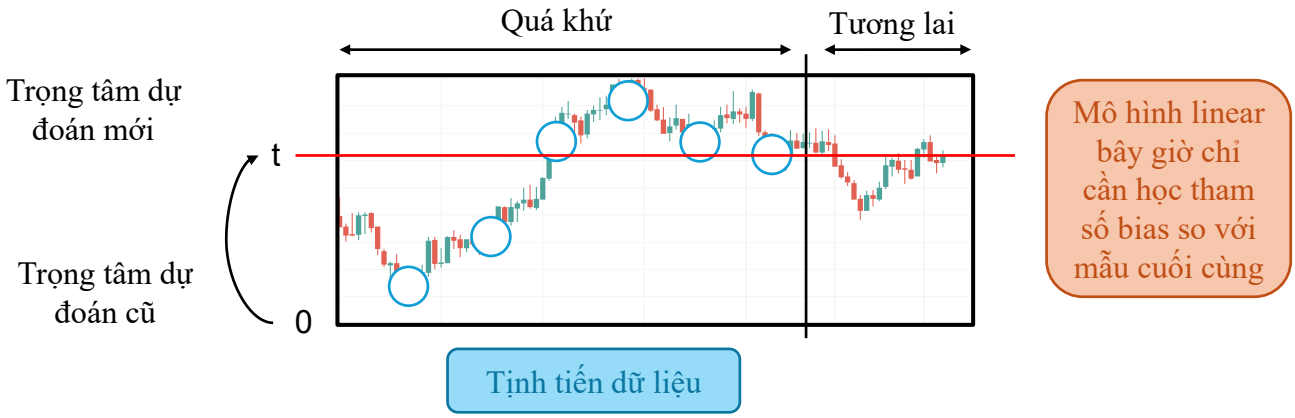
Trong đó (huấn luyện):

- T: số bước dự đoán (horizon); B: kích thước batch.
- $\hat{y}_t$, y_t: giá trị dự đoán và giá trị thực tại bước tương lai t.
- MSE: trung bình sai số bình phương trên toàn bộ T bước và B mẫu (tối ưu trực tiếp, không cuộn từng bước).

Độ phức tạp. Như Linear: tham số $\approx T \times L$ (cộng T bias); chi phí $\propto B \times T \times L$.



Hình 11: Quy trình chạy và huấn luyện của mô hình LTSF-NLinear.



Hình 12: Khi chúng ta biến đặt mốc cục bộ giá trị cuối cùng, mô hình Linear sẽ cần phải học ít thông tin hơn để đưa ra kết quả.

V. Mô hình LTSF-Linear: DLinear

Cơ chế: DLinear (đơn biến) *phân rã* chuỗi thành **trend** và **seasonal/residual**, sau đó *chiếu tuyến tính* riêng cho từng phần và *cộng* lại. Cách làm này phù hợp khi dữ liệu có trend và/hoặc seasonal rõ rệt.

Ví dụ về $\mathrm{MA}_m(\cdot)$ **(moving average-cách tạo trend):**

Giả sử m=3 với xử lý rìa kiểu `replicate` (tạm dịch là: lặp biên). Với $\mathbf{x} = [10, 12, 11, 13]$:

$$\mathrm{MA}_3(\mathbf{x}) = \left[\tfrac{10+10+12}{3},\ \tfrac{10+12+11}{3},\ \tfrac{12+11+13}{3},\ \tfrac{11+13+13}{3}\right] = [10.67,\ 11.00,\ 12.00,\ 12.33].$$

Chuỗi trên đóng vai trò *trend*; phần *seasonal/residual* là $\mathbf{x} - \mathrm{MA}_3(\mathbf{x})$.

Thuật ngữ:

- **trend**: thành phần thay đổi chậm theo thời gian, phản ánh xu hướng dài hạn của chuỗi.
- **seasonal/residual**: phần còn lại sau khi bỏ trend; thường chứa nhịp lặp theo chu kỳ và dao động nhanh.

Thiết lập dữ liệu. Với chuỗi $\{x_t\}$, tại thời điểm t tạo

$$\mathbf{x} = [x_{t-L+1}, \ldots, x_t] \in \mathbb{R}^L, \qquad \mathbf{y} = [x_{t+1}, \ldots, x_{t+T}] \in \mathbb{R}^T.$$

Trong đó (thiết lập):

- $\mathbf{x}$: input quá khứ dài L; $\mathbf{y}$: mục tiêu T bước tương lai.
- $\mathbf{1}_L, \mathbf{1}_T$: vectơ toàn 1 kích thước L và T.

Công thức. Chọn bậc làm mượt m (ví dụ m=5). Phân rã bằng moving average, dự đoán tuyến tính theo từng phần, rồi cộng:

$$\begin{aligned} x_t &= \mathrm{MA}_m(x) && \text{trend (moving average của } x) \\ x_s &= x - x_t && \text{seasonal/residual (phần còn lại)} \\ \hat{y}_t &= W_t\, x_t + b_t && \text{linear head cho trend } (W_t \in \mathbb{R}^{T\times L},\ b_t \in \mathbb{R}^T) \\ \hat{y}_s &= W_s\, x_s + b_s && \text{linear head cho seasonal/residual } (W_s \in \mathbb{R}^{T\times L},\ b_s \in \mathbb{R}^T) \\ \hat{y} &= \hat{y}_{trend} + \hat{y}_{seasonal} \in \mathbb{R}^T && \text{tổng hợp (forecast cuối cùng)} \end{aligned}$$

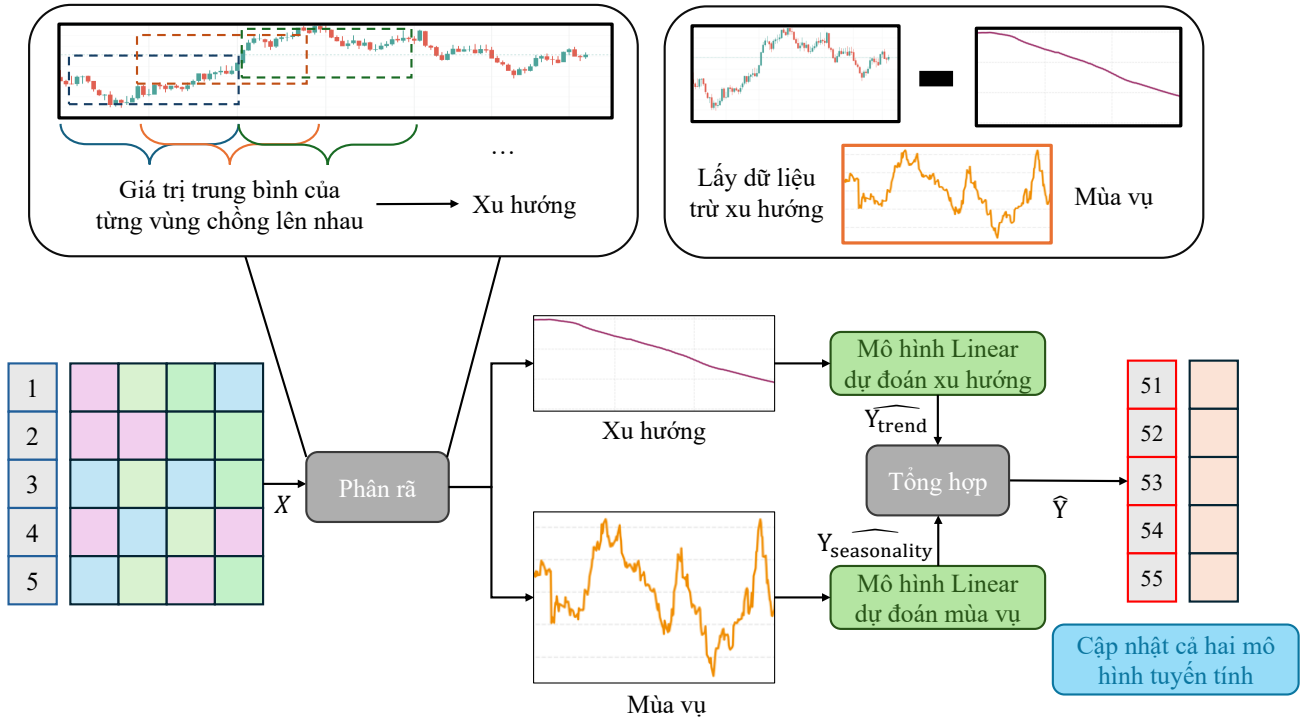
Huấn luyện (DMS). Tối ưu đồng thời cả T bước bằng MSE trên batch B:

$$\mathcal{L}_{\text{MSE}} = \frac{1}{B\,T} \sum_{n=1}^{B} \sum_{t=1}^{T} \left(\hat{y}_t - y_t\right)^2.$$

Trong đó (huấn luyện):

- T: số bước dự đoán (horizon); B: kích thước batch.
- $\hat{y}_t$, y_t: giá trị dự đoán và thực tại bước t.
- MSE: trung bình sai số bình phương trên toàn bộ T bước và B mẫu (tối ưu trực tiếp, không cuộn từng bước).

Độ phức tạp. Hai đầu tuyến tính nên tham số $\approx 2\,(T \times L)$ (cộng $2T$ bias); chi phí $\propto B \times T \times L$ (nhân đôi hằng số do hai nhánh), vẫn rất nhẹ.



Hình 13: Quy trình chạy và huấn luyện của mô hình LTSF-DLinear.

Lưu ý

Trong DLinear, tham số m biểu thị độ dài cửa sổ trung bình trượt (moving average window length), tức số điểm dữ liệu được lấy trung bình để ước lượng xu hướng Trend(x_t). Tham số này xác định mức độ làm mượt của *low-pass filter* — bộ lọc cho phép thành phần tần số thấp (xu hướng dài hạn) đi qua và loại bỏ thành phần tần số cao (dao động ngắn hạn). Khi m lớn, xu hướng trở nên trơn hơn nhưng làm giảm chi tiết chu kỳ; ngược lại, m nhỏ giúp phản ứng nhanh hơn với biến động ngắn hạn. DLinear phù hợp với chuỗi có xu hướng hoặc chu kỳ rõ, trong khi NLinear hiệu quả hơn với dữ liệu có *level shift*, tức là các điểm thời gian mà giá trị kỳ vọng (hoặc trung bình cục bộ) của chuỗi thay đổi đột ngột.

VI. Xây dựng các mô hình LTSF-Linear từ đầu (from sratch)

Để minh họa rõ cơ chế hoạt động, ta xây dựng lại ba mô hình **Linear**, **NLinear** và **DLinear** với phiên bản đơn giản, dễ hiểu hơn từ đầu.

Bước 1: Import thư viện và tạo dữ liệu mẫu

```python
import torch
import torch.nn as nn
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

data = torch.tensor([[100.0, 102.0, 98.0, 105.0, 103.0]])  # [1, 5]
print(f"Input: {data.squeeze().numpy()}")

target = torch.tensor([[101.0, 104.0, 106.0]])  # [1, 3]
print(f"Target: {target.squeeze().numpy()}")
```

Ở bước này, bạn đã có dữ liệu đầu vào $\mathbf{x} \in \mathbb{R}^{1\times 5}$ (5 bước quá khứ) và mục tiêu $\mathbf{y} \in \mathbb{R}^{1\times 3}$ (3 bước cần dự báo). Ta sẽ dùng chúng để kiểm tra mô hình ở các bước sau.

Bước 2: Xây dựng mô hình `Linear` với trọng số cố định

Code Exercise

```python
class Linear(nn.Module):
    def __init__(self):
        super().__init__()
        self.linear = nn.Linear(5, 3)
        # Set weights cụ thể
        with torch.no_grad():
            self.linear.weight.data = torch.tensor([
                [0.2, 0.3, -0.1, 0.4, 0.2],  # Output 1
                [0.1, 0.2, 0.3, 0.3, 0.1],   # Output 2
                [0.0, 0.1, 0.2, 0.4, 0.3]    # Output 3
            ])
            self.linear.bias.data = torch.tensor([1.0, 2.0, 3.0])

    def forward(self, x):
        return ***Your Code Here***

# Test Linear
linear_model = Linear()
```

```
linear_out = ***Your Code Here***

print("LINEAR MODEL:")
print(f"   Weights:\n{linear_model.linear.weight.data.numpy()}")
print(f"   Bias: {linear_model.linear.bias.data.numpy()}")
print(f"   Input: {data.squeeze().numpy()}")
print(f"   Output: {linear_out.squeeze().detach().numpy()}")

# Manual calculation để verify
print("   Manual calc:")
for i in range(3):
    w = linear_model.linear.weight.data[i]
    b = linear_model.linear.bias.data[i]
    result = torch.sum(w * data.squeeze()) + b
    print(f"     Out{i+1}: {result:.3f}")
```

Thực hành điền code vào các vị trí `***Your Code Here***` để hoàn thành các yêu cầu sau:

- Hoàn thiện hàm `forward` để đưa đầu vào qua lớp `nn.Linear` và nhận đầu ra có kích thước phù hợp.
- Gọi mô hình với `data` để lấy `linear_out`.
- Đối chiếu Output với phần tính tay (Manual calculation) để kiểm tra công thức tuyến tính $\hat{y}_i = \sum_j W_{ij} x_j + b_i$.

Bước 3: Xây dựng mô hình `DLinear` với trọng số cố định

Code Exercise

```
class DLinear(nn.Module):
    def __init__(self):
        super().__init__()
        self.linear_trend = nn.Linear(5, 3)
        self.linear_seasonal = nn.Linear(5, 3)

        # Set weights cụ thể cho trend
        with torch.no_grad():
            self.linear_trend.weight.data = torch.tensor([
                [0.3, 0.3, 0.2, 0.1, 0.1],
                [0.2, 0.3, 0.3, 0.1, 0.1],
                [0.1, 0.2, 0.3, 0.3, 0.1]
            ])
            self.linear_trend.bias.data = torch.tensor([0.5, 1.0, 1.5])

            # Set weights cho seasonal
            self.linear_seasonal.weight.data = torch.tensor([
                [0.1, -0.2, 0.3, -0.1, 0.0],
                [0.0, 0.1, -0.2, 0.2, 0.1],
                [-0.1, 0.0, 0.1, 0.1, 0.2]
```

```
            ])
            self.linear_seasonal.bias.data = torch.tensor([0.2, 0.3, 0.1])

    def decompose(self, x):
        # Simple moving average (window=3) - replicate padding
        x_np = x.squeeze().numpy()
        trend = []
        for i in range(len(x_np)):
            if i == 0:
                ***Your Code Here***
            elif i == len(x_np) - 1:
                ***Your Code Here***
            else:
                ***Your Code Here***
            trend.append(avg)

        trend_tensor = torch.tensor(trend).unsqueeze(0)
        seasonal = ***Your Code Here***
        return trend_tensor, seasonal

    def forward(self, x):
        trend, seasonal = ***Your Code Here***
        trend_pred = self.linear_trend(trend)
        seasonal_pred = self.linear_seasonal(seasonal)
        return ***Your Code Here***
# Test DLinear
dlinear_model = DLinear()
dlinear_out = dlinear_model(data)

print("DLINEAR MODEL:")
trend, seasonal = dlinear_model.decompose(data)
print(f"   Input: {data.squeeze().numpy()}")
print(f"   Trend: {trend.squeeze().numpy()}")
print(f"   Seasonal: {seasonal.squeeze().numpy()}")
print(f"   Output: {dlinear_out.squeeze().detach().numpy()}")

# Show component predictions
trend_pred = dlinear_model.linear_trend(trend)
seasonal_pred = dlinear_model.linear_seasonal(seasonal)
print(f"   Trend pred: {trend_pred.squeeze().detach().numpy()}")
print(f"   Seasonal pred: {seasonal_pred.squeeze().detach().numpy()}")
```

Thực hành điền code vào các vị trí `***Your Code Here***` để hoàn thành các yêu cầu sau:

- Trong `decompose`: áp dụng trung bình trượt cửa sổ 3 điểm với replicate padding ở hai đầu để tạo dãy trend; sau đó lấy tín hiệu gốc trừ trend tại từng vị trí để thu được seasonal.
- Trong `forward`: gọi `decompose` để lấy trend và seasonal; đưa trend qua `linear_trend`, đưa seasonal qua `linear_seasonal`; cộng hai kết quả để tạo đầu ra cuối cùng.
- Chạy khối kiểm thử để quan sát lần lượt: Input, trend, seasonal, dự đoán từ nhánh trend, dự đoán từ nhánh seasonal và Output tổng hợp.

Mục tiêu học tập:

- Hiểu quy trình tách chuỗi bằng thao tác làm mượt rồi trừ: trước hết lấy trend bằng trung bình trượt, sau đó trừ trend khỏi tín hiệu gốc để có phần còn lại.
- Nắm cách hai nhánh tuyến tính xử lý riêng hai phần và được cộng lại để tạo dự báo, qua đó thấy được tính tách biệt và khả năng diễn giải của DLinear.

Bước 4: Xây dựng mô hình `NLinear` với trọng số cố định

Code Exercise

```python
class NLinear(nn.Module):
    def __init__(self):
        super().__init__()
        self.linear = nn.Linear(5, 3)
        # Set weights cụ thể
        with torch.no_grad():
            self.linear.weight.data = torch.tensor([
                [0.1, 0.2, 0.1, 0.3, 0.3],
                [0.2, 0.1, 0.3, 0.2, 0.2],
                [0.3, 0.2, 0.2, 0.2, 0.1]
            ])
            self.linear.bias.data = torch.tensor([0.5, 1.0, 2.0])

    def forward(self, x):
        # Normalize bằng last value
        last_value = ***Your Code Here***
        x_norm = ***Your Code Here***
        pred_norm = ***Your Code Here***
        pred = ***Your Code Here***
        return pred

# Test NLinear
nlinear_model = NLinear()
nlinear_out = nlinear_model(data)

print("NLINEAR MODEL:")
last_val = data[:, -1:]
x_norm = data - last_val
pred_norm = nlinear_model.linear(x_norm)

print(f"   Input: {data.squeeze().numpy()}")
print(f"   Last value: {last_val.squeeze().item()}")
print(f"   Normalized: {x_norm.squeeze().numpy()}")
print(f"   Pred normalized: {pred_norm.squeeze().detach().numpy()}")
print(f"   Final output: {nlinear_out.squeeze().detach().numpy()}")

# Manual verification
print("   Manual calc:")
for i in range(3):
    w = nlinear_model.linear.weight.data[i]
    b = nlinear_model.linear.bias.data[i]
```

```
    norm_pred = torch.sum(w * x_norm.squeeze()) + b
    final = norm_pred + last_val.squeeze()
    print(f"     Out{i+1}: {norm_pred:.3f} + {last_val.squeeze().item():.1f} = {final:.
                                                3f}")
```

Thực hành điền code vào các vị trí `***Your Code Here***` để hoàn thành các yêu cầu sau:

- Điền last_value là cột cuối của cửa sổ đầu vào theo trục thời gian.
- Chuẩn hoá đầu vào bằng cách trừ last_value khỏi toàn bộ dãy để thu được x_norm.
- Tính pred_norm bằng cách đưa x_norm qua lớp tuyến tính đã khai báo.
- Cộng lại mốc last_value vào pred_norm để có pred và trả về pred.

Mục tiêu học tập:

- Hiểu cơ chế tái căn tâm theo giá trị cuối cùng: mô hình học trên độ lệch so với mốc hiện tại, sau đó cộng mốc để quay về thang gốc.
- Quan sát chuỗi bước in ra: giá trị cuối, đầu vào đã chuẩn hoá, dự đoán trên không gian chuẩn hoá, và đầu ra cuối cùng.

VII. Bài toán dự đoán giá cổ phiếu sử dụng LTSF-Linear Models

Trong phần này, ta áp dụng họ mô hình **LTSF-Linear** (Linear, NLinear, DLinear) cho bài toán **dự đoán giá đóng cửa cổ phiếu VIC trong 7 ngày tới**. Mỗi mô hình được huấn luyện với các độ dài cửa sổ quá khứ khác nhau — 7, 30, 120 và 480 ngày — nhằm đánh giá khả năng học và khái quát trên nhiều quy mô thời gian. Kết quả dự đoán sẽ được so sánh với nhau để kiểm chứng hiệu quả và độ ổn định của các phương pháp tuyến tính trong dự báo chuỗi tài chính.

Bước 1: Import thư viện và tải dữ liệu

```
!gdown 18J_Z8b-qMMj9wm5eGyQ-1nPS16PfRePK

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import PercentFormatter
import matplotlib.dates as mdates

import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

```
12
13 import torch
14 import torch.nn as nn
15 from torch.utils.data import Dataset, DataLoader
16 from sklearn.metrics import mean_squared_error,
17 mean_absolute_error, r2_score
18 from sklearn.preprocessing import StandardScaler
19
20 plt.style.use("seaborn-v0_8")
21 sns.set_palette("husl")
22 print(f"Device available: {"CUDA" if torch.cuda.is_available() else "CPU"}")
23
24 # Đọc dữ liệu
25 df = pd.read_csv("VIC.csv")
26 print(f"Dataset shape: {df.shape}")
27 print(f"Date range: {df["time"].min()} to {df["time"].max()}")
28
29 # Xử lý và sắp xếp theo thời gian
30 df["time"] = pd.to_datetime(df["time"])
31 df = df.sort_values("time").reset_index(drop=True)
32
33 # Thông tin cơ bản
34 print("\nFirst few rows:")
35 display(df.head())
36
37 print("\nData types:")
38 print(df.dtypes)
39
40 print("\nBasic statistics:")
41 display(df.describe())
```

Trong bước đầu tiên, các thư viện cần thiết được import, bao gồm: `pandas`, `numpy`, `matplotlib`, `seaborn` cho xử lý và trực quan hóa dữ liệu; `torch` và `sklearn` cho xây dựng mô hình và đánh giá hiệu suất. Tập dữ liệu **VIC.csv** được tải về và đọc bằng `pandas.read_csv()`, sau đó cột `time` được chuyển sang kiểu `datetime` và sắp xếp theo thứ tự thời gian.

Cuối cùng, dữ liệu được kiểm tra sơ bộ qua kích thước tập, khoảng thời gian, kiểu dữ liệu và thống kê mô tả cơ bản — giúp xác nhận cấu trúc dữ liệu đầu vào trước khi xử lý và huấn luyện mô hình.

	time	open	high	low	close	volume	symbol
0	2020-08-03	77.33	78.31	75.56	77.87	164310	VIC
1	2020-08-04	78.84	78.84	77.69	78.22	229230	VIC
2	2020-08-05	78.22	78.67	75.38	77.33	434490	VIC
3	2020-08-06	78.22	78.40	77.42	77.78	332340	VIC
4	2020-08-07	77.87	78.40	77.60	77.78	182500	VIC

Hình 14: Vài samples của tập dữ liệu VIC

Bước 2: Khám phá và tiền xử lý dữ liệu

```python
# Tạo essential features
print("1. Tạo essential features:")
df["daily_return"] = df["close"].pct_change()
df["close_log"] = np.log(df["close"])
print(" - daily_return: phần trăm thay đổi hàng ngày")
print(" - close_log: log giá đóng cửa")

# Kiểm tra missing values cho TẤT CẢ các cột
missing_info = df.isnull().sum()

# Chỉ xử lý giá trị khuyết cần thiết
if df["daily_return"].isnull().sum() > 0:
    df["daily_return"].fillna(0, inplace=True)

# Kiểm tra lại sau khi xử lý
missing_after = df.isnull().sum()
missing_after
```

Trong bước này, hai đặc trưng cơ bản được tạo ra từ giá đóng cửa:

- `daily_return`: phần trăm thay đổi giá giữa hai ngày liên tiếp, phản ánh biến động ngắn hạn.
- `close_log`: logarit tự nhiên của giá đóng cửa, giúp giảm độ lệch và ổn định phân phối dữ liệu.

Sau đó, tập dữ liệu được kiểm tra để phát hiện giá trị khuyết (*missing values*) trên toàn bộ các cột. Nếu xuất hiện giá trị `NaN` ở cột `daily_return` (do phép tính phần trăm thay đổi tại dòng đầu tiên), ta thay thế bằng 0 để đảm bảo tính toàn vẹn dữ liệu trước khi đưa vào huấn luyện.

Việc kiểm tra lại sau khi xử lý giúp xác nhận rằng toàn bộ dữ liệu hiện đã hoàn chỉnh, sẵn sàng cho các bước tiền xử lý và tạo tập huấn luyện tiếp theo.

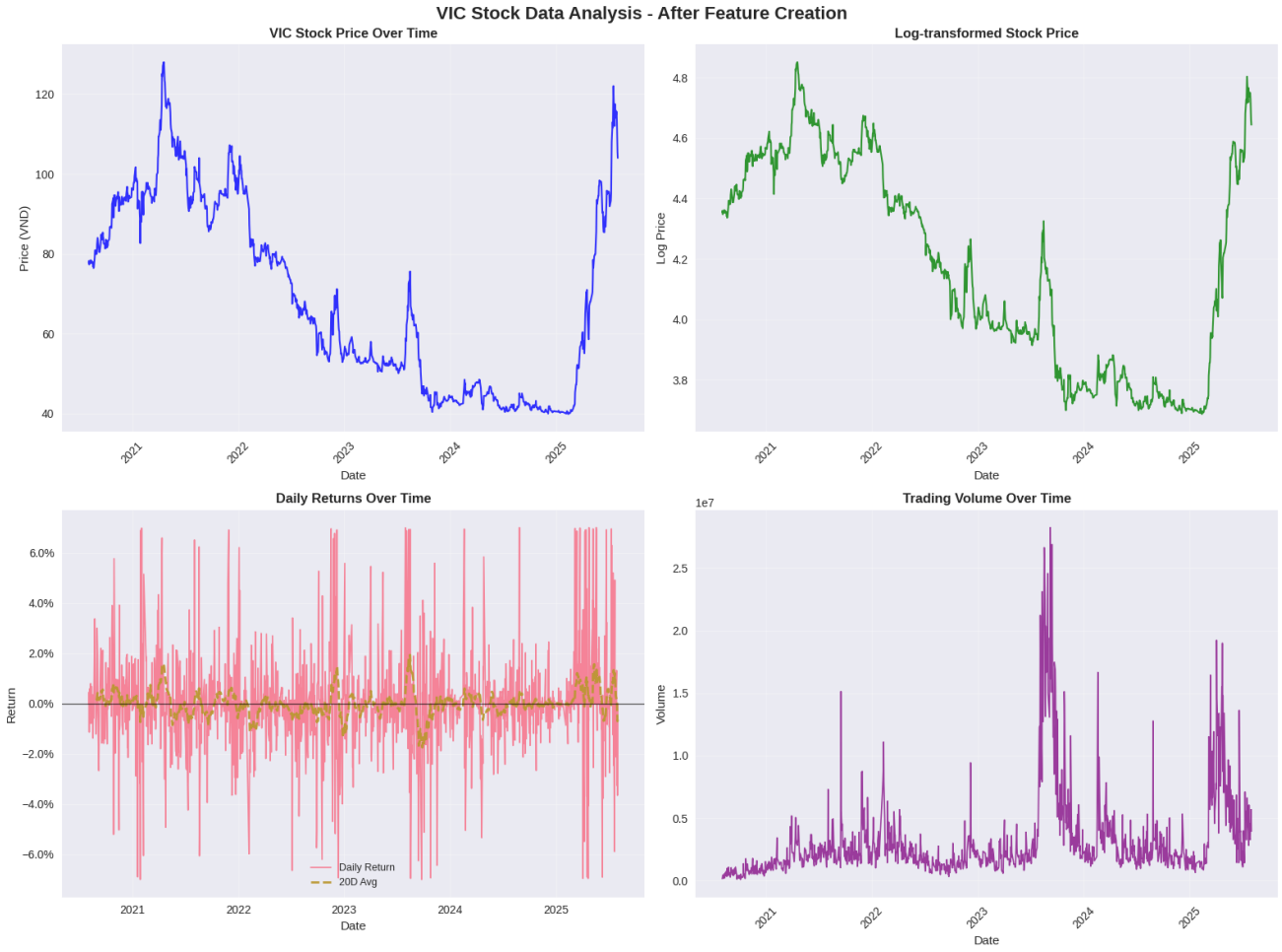
```python
fig, axes = plt.subplots(2, 2, figsize=(16, 12))
fig.suptitle("VIC Stock Data Analysis - After Feature Creation", fontsize=16,
                                                fontweight="bold")

# 1) Stock Price Over Time
axes[0, 0].plot(df["time"], df["close"], linewidth=1.5, color="blue", alpha=0.8)
axes[0, 0].set_title("VIC Stock Price Over Time", fontsize=12, fontweight="bold")
axes[0, 0].set_xlabel("Date")
axes[0, 0].set_ylabel("Price (VND)")
axes[0, 0].grid(True, alpha=0.3)
axes[0, 0].tick_params(axis="x", rotation=45)

# 2) Log-transformed Price
axes[0, 1].plot(df["time"], df["close_log"], linewidth=1.5, color="green", alpha=0.8)
axes[0, 1].set_title("Log-transformed Stock Price", fontsize=12, fontweight="bold")
```

```
axes[0, 1].set_xlabel("Date")
axes[0, 1].set_ylabel("Log Price")
axes[0, 1].grid(True, alpha=0.3)
axes[0, 1].tick_params(axis="x", rotation=45)

# 3) Daily Returns Over Time
ret_ts = df[["time", "daily_return"]].dropna().copy()
ret_ts["ret_ma20"] = ret_ts["daily_return"].rolling(20).mean()

axes[1, 0].plot(ret_ts["time"], ret_ts["daily_return"], linewidth=1.2, alpha=0.85,
                                        label="Daily Return")
axes[1, 0].plot(ret_ts["time"], ret_ts["ret_ma20"], linewidth=2.0, linestyle="--",
                                        alpha=0.95, label="20D Avg")
axes[1, 0].axhline(0, color="black", linewidth=1, alpha=0.6)
axes[1, 0].set_title("Daily Returns Over Time", fontsize=12, fontweight="bold")
axes[1, 0].set_xlabel("Date")
axes[1, 0].set_ylabel("Return")
axes[1, 0].yaxis.set_major_formatter(PercentFormatter(1.0))
axes[1, 0].xaxis.set_major_locator(mdates.AutoDateLocator())
axes[1, 0].xaxis.set_major_formatter(mdates.ConciseDateFormatter(mdates.AutoDateLocator
                                        ()))
axes[1, 0].tick_params(axis="x", rotation=0)
axes[1, 0].grid(True, alpha=0.3)
axes[1, 0].legend(fontsize=9)

# 4) Volume Over Time
axes[1, 1].plot(df["time"], df["volume"], linewidth=1.2, color="purple", alpha=0.75)
axes[1, 1].set_title("Trading Volume Over Time", fontsize=12, fontweight="bold")
axes[1, 1].set_xlabel("Date")
axes[1, 1].set_ylabel("Volume")
axes[1, 1].grid(True, alpha=0.3)
axes[1, 1].tick_params(axis="x", rotation=45)

plt.tight_layout()
plt.show()
```

Sau khi xử lý và bổ sung đặc trưng, dữ liệu được trực quan hoá để quan sát xu hướng và cấu trúc biến động theo thời gian:

- **(1) Stock Price Over Time:** biểu diễn giá đóng cửa theo thời gian, thể hiện biến động tổng thể của cổ phiếu VIC.
- **(2) Log-transformed Price:** đồ thị giá đã logarit hoá, giúp làm phẳng các dao động lớn, dễ quan sát xu hướng dài hạn.
- **(3) Daily Returns Over Time:** hiển thị phần trăm thay đổi giá hằng ngày cùng đường trung bình trượt 20 ngày, giúp nhận diện giai đoạn biến động mạnh hay ổn định.
- **(4) Trading Volume Over Time:** biểu đồ khối lượng giao dịch, phản ánh mức độ thanh khoản và sự quan tâm của thị trường.

Những biểu đồ này cung cấp cái nhìn trực quan đầu tiên về hành vi của cổ phiếu VIC — bao gồm xu hướng giá, độ biến động và mối liên hệ giữa giá và khối lượng giao dịch — là nền tảng quan trọng trước khi trích xuất đặc trưng đầu vào cho mô hình dự báo.



Hình 15: Các biểu đồ cung cấp cái nhìn trực quan về hành vi của cổ phiếu VIC

Bước 3: Chuẩn bị dữ liệu chuỗi thời gian đơn biến cho mô hình dự báo

```python
class UnivariateTimeSeriesDataset(Dataset):
    """Dataset for univariate time series forecasting - 7 days ahead"""
    def __init__(self, data, seq_len, pred_len=7, target_col="close", normalize=False):
        self.data = data.dropna().reset_index(drop=True)
        self.seq_len = seq_len
        self.pred_len = pred_len
        self.target_col = target_col
        self.normalize = normalize

        self.series = self.data[target_col].values

```

```
        if normalize:
            self.mean = np.mean(self.series)
            self.std = np.std(self.series)

    def __len__(self):
        return len(self.series) - self.seq_len - self.pred_len + 1

    def __getitem__(self, idx):
        x = self.series[idx:idx+self.seq_len].copy()
        y = self.series[idx+self.seq_len:idx+self.seq_len+self.pred_len].copy()

        if self.normalize:
            x = (x - self.mean) / self.std
            y = (y - self.mean) / self.std

        return torch.FloatTensor(x), torch.FloatTensor(y)

    def denormalize(self, normalized_values):
        if not self.normalize:
            return normalized_values
        return normalized_values * self.std + self.mean


def create_univariate_datasets(df, seq_lengths, pred_len=7, target_col="close"):
    """Create univariate datasets for different sequence lengths"""
    datasets = {}

    for seq_len in seq_lengths:
        dataset = UnivariateTimeSeriesDataset(
            data=df, seq_len=seq_len, pred_len=pred_len,
            target_col=target_col, normalize=False
        )
        datasets[f"{seq_len}d"] = dataset

    return datasets


# Create datasets for 7-day prediction with new input lengths
seq_lengths = [7, 30, 120, 480]  # Input sequence lengths
pred_len = 7  # Predict 7 days ahead

# Create datasets using log prices for better stability
datasets = create_univariate_datasets(df, seq_lengths, pred_len, "close_log")

print(f"Created datasets for 7-day prediction with input lengths: {seq_lengths}")
for name, dataset in datasets.items():
    print(f"- {name}: {len(dataset)} samples")

print("Using log-transformed prices for stability.")
```

Lớp `UnivariateTimeSeriesDataset` được định nghĩa để tạo tập dữ liệu dự báo đơn biến. Mỗi mẫu gồm hai phần: $\mathbf{x} \in \mathbb{R}^L$ (chuỗi đầu vào gồm L ngày trước) và $\mathbf{y} \in \mathbb{R}^T$ (giá trị mục tiêu cho T

ngày tiếp theo). Ở đây, dữ liệu sử dụng giá đóng cửa logarit (`close_log`) để tăng tính ổn định khi huấn luyện mô hình.

Các tập dữ liệu được tạo cho bốn độ dài cửa sổ đầu vào: **7**, **30**, **120** và **480** ngày, với mục tiêu dự báo **7 ngày tiếp theo**.

```python
def create_time_based_splits(dataset, train_ratio=0.7, val_ratio=0.15):
    """Create time-based splits for time series data"""
    total_len = len(dataset)
    train_len = int(total_len * train_ratio)
    val_len = int(total_len * val_ratio)

    # Time-based splits (no shuffling to preserve temporal order)
    train_indices = list(range(0, train_len))
    val_indices = list(range(train_len, train_len + val_len))
    test_indices = list(range(train_len + val_len, total_len))

    train_dataset = torch.utils.data.Subset(dataset, train_indices)
    val_dataset = torch.utils.data.Subset(dataset, val_indices)
    test_dataset = torch.utils.data.Subset(dataset, test_indices)

    return train_dataset, val_dataset, test_dataset


# Create splits for all sequence lengths
data_splits = {}

print("Creating time-based data splits:")
print("- Train: 70%, Validation: 15%, Test: 15%")
print("- Temporal order preserved (no shuffling)")

for seq_name, dataset in datasets.items():
    train, val, test = create_time_based_splits(dataset)
    data_splits[seq_name] = {
        "train": train,
        "val": val,
        "test": test
    }

print(f"\nTotal datasets with splits: {len(data_splits)}")

# Verify split integrity
print(f"\n=== SPLIT VERIFICATION ===")
for seq_name, splits in data_splits.items():
    total_samples = len(splits["train"]) + len(splits["val"]) + len(splits["test"])
    original_samples = len(datasets[seq_name])

    print(f"{seq_name}: {total_samples}/{original_samples} samples"
          if total_samples == original_samples
          else f"{seq_name}: ERROR - {total_samples}/{original_samples}")

print("\nData splits created successfully!")
```

Sau khi tạo tập dữ liệu, dữ liệu được chia theo tỷ lệ thời gian cố định: **70%** cho huấn luyện, **15%** cho xác thực và **15%** cho kiểm thử. Không áp dụng xáo trộn (*no shuffling*) để giữ nguyên thứ tự thời gian. Cách chia này đảm bảo tính liên tục của chuỗi và phản ánh đúng bối cảnh dự báo thực tế, đồng thời giúp đánh giá khả năng mô hình tổng quát hoá qua các giai đoạn khác nhau.

```
Creating time-based data splits:
- Train: 70%, Validation: 15%, Test: 15%
- Temporal order preserved (no shuffling)

Total datasets with splits: 4

=== SPLIT VERIFICATION ===
7d: 1487/1487 samples ✓
30d: 1464/1464 samples ✓
120d: 1374/1374 samples ✓
480d: 1014/1014 samples ✓

Data splits created successfully!
```

Hình 16: Tổng quan dữ liệu sau khi được chia thành các tập

Bước 4: Xây dựng và khởi tạo các mô hình LTSF-Linear

Code Exercise

```python
# LTSF-Linear Models Implementation for Univariate Time Series (7-day prediction)
class Linear(nn.Module):
    """Simple Linear model for univariate time series forecasting"""
    def __init__(self, seq_len, pred_len=7):
        super(Linear, self).__init__()
        self.seq_len = seq_len
        self.pred_len = pred_len
        self.linear = ***Your Code Here***

    def forward(self, x):
        # x: [batch_size, seq_len] - historical prices
        # Output: [batch_size, pred_len] - 7-day future predictions
        return ***Your Code Here***

class DLinear(nn.Module):
    """Decomposition Linear for univariate time series - handles trend and seasonality
                                                    """
    def __init__(self, seq_len, pred_len=7, moving_avg=5):
        super(DLinear, self).__init__()
        self.seq_len = seq_len
        self.pred_len = pred_len
        self.moving_avg = min(moving_avg, seq_len - 1)

        # Linear layers for trend and seasonal components
        self.linear_trend = ***Your Code Here***
        self.linear_seasonal = ***Your Code Here***

        # Create moving average kernel for trend extraction
```

```
        self.register_buffer('avg_kernel', torch.ones(1, 1, self.moving_avg) / self.
                                            moving_avg)

    def decompose(self, x):
        """Decompose series into trend and seasonal components"""
        batch_size, seq_len = x.shape
        x_reshaped = x.unsqueeze(1)

        # Apply moving average for trend
        padding = self.moving_avg // 2
        x_padded = torch.nn.functional.pad(x_reshaped, (padding, padding), mode='
                                            replicate')
        trend = torch.nn.functional.conv1d(x_padded, self.avg_kernel, padding=0)
        trend = trend.squeeze(1)

        # Ensure trend has same length as input
        if trend.shape[1] != seq_len:
            trend = torch.nn.functional.interpolate(
                trend.unsqueeze(1), size=seq_len, mode='linear', align_corners=False
            ).squeeze(1)

        seasonal = ***Your Code Here***
        return ***Your Code Here***

    def forward(self, x):
        trend, seasonal = ***Your Code Here***
        trend_pred = ***Your Code Here***
        seasonal_pred = ***Your Code Here***
        return ***Your Code Here***

class NLinear(nn.Module):
    """Normalized Linear for univariate time series - handles distribution shift"""
    def __init__(self, seq_len, pred_len=7):
        super(NLinear, self).__init__()
        self.seq_len = seq_len
        self.pred_len = pred_len
        self.linear = ***Your Code Here***

    def forward(self, x):
        # Normalize by subtracting last value
        last_value = ***Your Code Here***
        x_normalized = ***Your Code Here***
        pred_normalized = ***Your Code Here***
        pred = ***Your Code Here***
        return ***Your Code Here***

print("LTSF-Linear models implemented for 7-day univariate forecasting.")
```

Thực hành điền code vào các vị trí `***Your Code Here***` để hoàn thành các yêu cầu sau:

- Linear: khởi tạo lớp $[seq\_len] \to [pred\_len]$; trong forward, đưa x qua lớp đó để nhận đầu ra $[batch, pred\_len]$.

- DLinear: tạo hai lớp cho trend và seasonal; trong decompose, tính seasonal bằng $x - trend$ và trả về (trend, seasonal); trong forward, dự đoán từng nhánh rồi cộng.
- NLinear: lấy last_value $= x[:, -1:]$, chuẩn hoá $x -$ last_value, dự đoán trên không gian chuẩn hoá, cộng lại mốc để ra kết quả.

Mục tiêu học tập:

- Nắm đường đi dữ liệu và kích thước đầu vào–đầu ra ở cả ba biến thể.
- Hiểu cơ chế tách–cộng của DLinear và tái căn tâm theo giá trị cuối của NLinear.
- Giữ nguyên kích thước batch và trục thời gian; tránh dùng `squeeze()` gây mất chiều.

Sau đó ta tạo cấu hình và khởi tạo mô hình:

```
# Model Configuration and Creation for 7-day Prediction
horizon_configs = {
    '7d': {'seq_len': 7, 'pred_len': 7},
    '30d': {'seq_len': 30, 'pred_len': 7},
    '120d': {'seq_len': 120, 'pred_len': 7},
    '480d': {'seq_len': 480, 'pred_len': 7}
}

# Create model instances for each combination
model_configs = {
    'Linear': {},
    'DLinear': {},
    'NLinear': {}
}

print("Creating model instances for 7-day prediction:")
for horizon, config in horizon_configs.items():
    seq_len = config['seq_len']
    pred_len = config['pred_len']

    # Create model instances
    model_configs['Linear'][horizon] = {
        'model': Linear(seq_len, pred_len),
        'seq_len': seq_len,
        'pred_len': pred_len
    }

    model_configs['DLinear'][horizon] = {
        'model': DLinear(seq_len, pred_len),
        'seq_len': seq_len,
        'pred_len': pred_len
    }

    model_configs['NLinear'][horizon] = {
        'model': NLinear(seq_len, pred_len),
        'seq_len': seq_len,
        'pred_len': pred_len
```

```
    }

    print(f"  {horizon}: seq_len={seq_len}, pred_len={pred_len}")
```

Đoạn mã trên định nghĩa bốn cấu hình độ dài cửa sổ đầu vào (7, 30, 120, 480 ngày) cùng chung đích dự báo 7 ngày, rồi khởi tạo ba mô hình Linear, DLinear và NLinear cho từng cấu hình và lưu vào `model_configs` để phục vụ huấn luyện và so sánh hiệu suất.

Bước 5: Huấn luyện và trực quan hoá trực loss

Đầu tiên ta tạo dataset chuẩn hoá dùng chung một scaler, chia tập theo thời gian và đánh giá trên thang giá gốc.

```
# Training Configuration with Data Normalization for 7-day Prediction
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# Create normalized dataset class
class NormalizedDataset(Dataset):
    def __init__(self, original_dataset, scaler=None):
        self.original_dataset = original_dataset

        # Fit scaler if not provided
        if scaler is None:
            all_data = []
            for i in range(len(original_dataset)):
                x, y = original_dataset[i]
                all_data.extend(x.numpy())
                all_data.extend(y.numpy())

            self.scaler = StandardScaler()
            self.scaler.fit(np.array(all_data).reshape(-1, 1))
        else:
            self.scaler = scaler

    def __len__(self):
        return len(self.original_dataset)

    def __getitem__(self, idx):
        x, y = self.original_dataset[idx]

        # Normalize features and targets
        x_norm = self.scaler.transform(x.numpy().reshape(-1, 1)).flatten()
        y_norm = self.scaler.transform(y.numpy().reshape(-1, 1)).flatten()

        return torch.FloatTensor(x_norm), torch.FloatTensor(y_norm)

    def denormalize(self, normalized_values):
        """Convert normalized values back to original scale"""
```

```
        return self.scaler.inverse_transform(normalized_values.reshape(-1, 1)).flatten()

# Create normalized datasets
normalized_datasets = {}
scaler = None

for horizon, dataset in datasets.items():
    normalized_dataset = NormalizedDataset(dataset, scaler)
    if scaler is None:
        scaler = normalized_dataset.scaler  # Use same scaler for all
    normalized_datasets[horizon] = normalized_dataset

# Create normalized splits
normalized_data_splits = {}
for horizon in datasets.keys():
    total_len = len(normalized_datasets[horizon])
    train_len = int(total_len * 0.7)
    val_len = int(total_len * 0.15)

    train_indices = list(range(0, train_len))
    val_indices = list(range(train_len, train_len + val_len))
    test_indices = list(range(train_len + val_len, total_len))

    normalized_data_splits[horizon] = {
        'train': torch.utils.data.Subset(normalized_datasets[horizon], train_indices),
        'val': torch.utils.data.Subset(normalized_datasets[horizon], val_indices),
        'test': torch.utils.data.Subset(normalized_datasets[horizon], test_indices)
    }

# Updated evaluation function for normalized data
def evaluate_model_normalized(model, test_loader, scaler, device='cpu'):
    """Evaluate model performance with denormalization"""
    model.eval()
    predictions = []
    actuals = []

    with torch.no_grad():
        for batch_x, batch_y in test_loader:
            batch_x, batch_y = batch_x.to(device), batch_y.to(device)
            outputs = model(batch_x)

            # Denormalize predictions and actuals
            for i in range(outputs.shape[0]):
                pred_denorm = scaler.inverse_transform(outputs[i].cpu().numpy().reshape(-1, 1)).flatten()
                actual_denorm = scaler.inverse_transform(batch_y[i].cpu().numpy().reshape(-1, 1)).flatten()
                predictions.extend(pred_denorm)
                actuals.extend(actual_denorm)

    predictions = np.array(predictions)
    actuals = np.array(actuals)

```

```
    # Calculate metrics on original scale
    mse = mean_squared_error(actuals, predictions)
    mae = mean_absolute_error(actuals, predictions)
    rmse = np.sqrt(mse)

    try:
        r2 = r2_score(actuals, predictions)
    except:
        r2 = -999

    return {
        'mse': mse,
        'mae': mae,
        'rmse': rmse,
        'r2': r2,
        'predictions': predictions,
        'actuals': actuals
    }
```

Lớp `NormalizedDataset` chuẩn hoá cả đầu vào và mục tiêu bằng cùng một `StandardScaler` dùng chung cho mọi horizon; tạo các tập train/val/test theo thứ tự thời gian; hàm `evaluate_model_normalized` suy luận trên dữ liệu chuẩn hoá, rồi đảo chuẩn hoá dự đoán và nhãn để tính các chỉ số (MSE, MAE, RMSE, R^2) trên thang giá gốc.

```
def train_model(model, train_loader, val_loader, num_epochs=50, lr=0.001, device="cpu")
                                                    :
    """Train one model and return epoch-wise train/val MSE (no printing)."""
    criterion = nn.MSELoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=lr)
    scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=20, gamma=0.5)

    model.to(device)
    train_losses, val_losses = [], []

    for _ in range(num_epochs):
        # Train
        model.train()
        epoch_train = 0.0
        for batch_x, batch_y in train_loader:
            batch_x, batch_y = batch_x.to(device), batch_y.to(device)
            optimizer.zero_grad()
            preds = model(batch_x)
            loss = criterion(preds, batch_y)
            loss.backward()
            optimizer.step()
            epoch_train += loss.item()
        train_losses.append(epoch_train / max(1, len(train_loader)))

        # Validate
        model.eval()
        epoch_val = 0.0
```

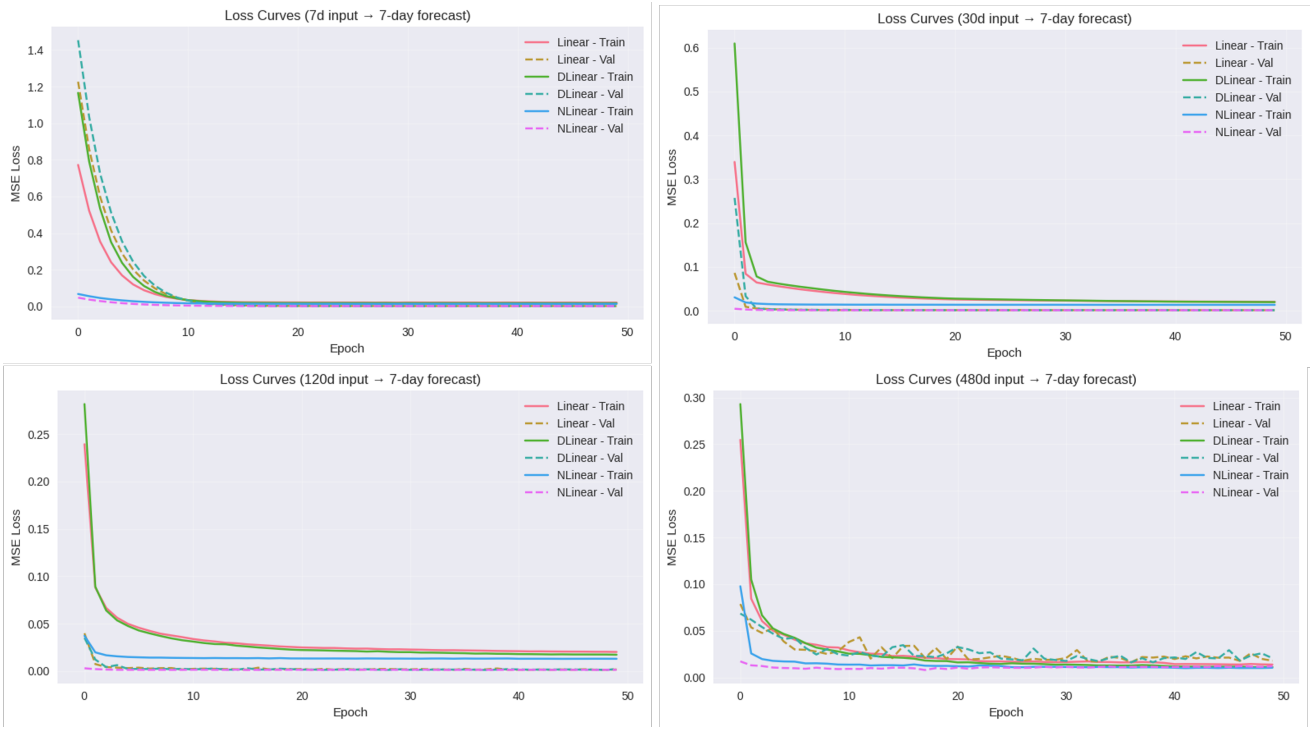
```
        with torch.no_grad():
            for batch_x, batch_y in val_loader:
                batch_x, batch_y = batch_x.to(device), batch_y.to(device)
                preds = model(batch_x)
                epoch_val += criterion(preds, batch_y).item()
        val_losses.append(epoch_val / max(1, len(val_loader)))

        scheduler.step()

    return model, train_losses, val_losses
```

Hàm `train_model` huấn luyện mô hình và trả về lịch sử `train_losses` / `val_losses` cho việc vẽ đồ thị, *không* in log theo từng epoch.

```
import matplotlib.pyplot as plt

def plot_loss_curves(loss_history, horizons):
    """Plot train/val losses for all models across horizons."""
    for hz in horizons:
        plt.figure(figsize=(8, 4.5))
        for model_name in ["Linear", "DLinear", "NLinear"]:
            if hz in loss_history.get(model_name, {}):
                tl = loss_history[model_name][hz]["train"]
                vl = loss_history[model_name][hz]["val"]
                plt.plot(tl, label=f"{model_name} - Train")
                plt.plot(vl, linestyle="--", label=f"{model_name} - Val")
        plt.title(f"Loss Curves ({hz} input  7-day forecast)")
        plt.xlabel("Epoch")
        plt.ylabel("MSE Loss")
        plt.grid(True, alpha=0.3)
        plt.legend()
        plt.tight_layout()
        plt.show()

batch_size = 32
num_epochs = 50
learning_rate = 0.001

results = {"Linear": {}, "DLinear": {}, "NLinear": {}}
trained_models = {"Linear": {}, "DLinear": {}, "NLinear": {}}
loss_history = {"Linear": {}, "DLinear": {}, "NLinear": {}}

for horizon in ["7d", "30d", "120d", "480d"]:
    train_loader = DataLoader(
        normalized_data_splits[horizon]["train"],
        batch_size=batch_size, shuffle=True, drop_last=True
    )
    val_loader = DataLoader(
        normalized_data_splits[horizon]["val"],
        batch_size=batch_size, shuffle=False, drop_last=True
```

```
    )
    test_loader = DataLoader(
        normalized_data_splits[horizon]["test"],
        batch_size=batch_size, shuffle=False, drop_last=True
    )

    for model_name in ["Linear", "DLinear", "NLinear"]:
        model = model_configs[model_name][horizon]["model"]
        trained_model, tr_losses, va_losses = train_model(
            model, train_loader, val_loader,
            num_epochs=num_epochs, lr=learning_rate, device=device
        )
        loss_history[model_name][horizon] = {"train": tr_losses, "val": va_losses}
        trained_models[model_name][horizon] = trained_model

        # (Tuỳ chọn) lưu kết quả test để tổng hợp cuối cùng, không in
        test_results = evaluate_model_normalized(trained_model, test_loader, scaler,
                                                 device)
        results[model_name][horizon] = test_results

# Chỉ trực quan hoá (không in log)
plot_loss_curves(loss_history, horizons=["7d", "30d", "120d", "480d"])
```

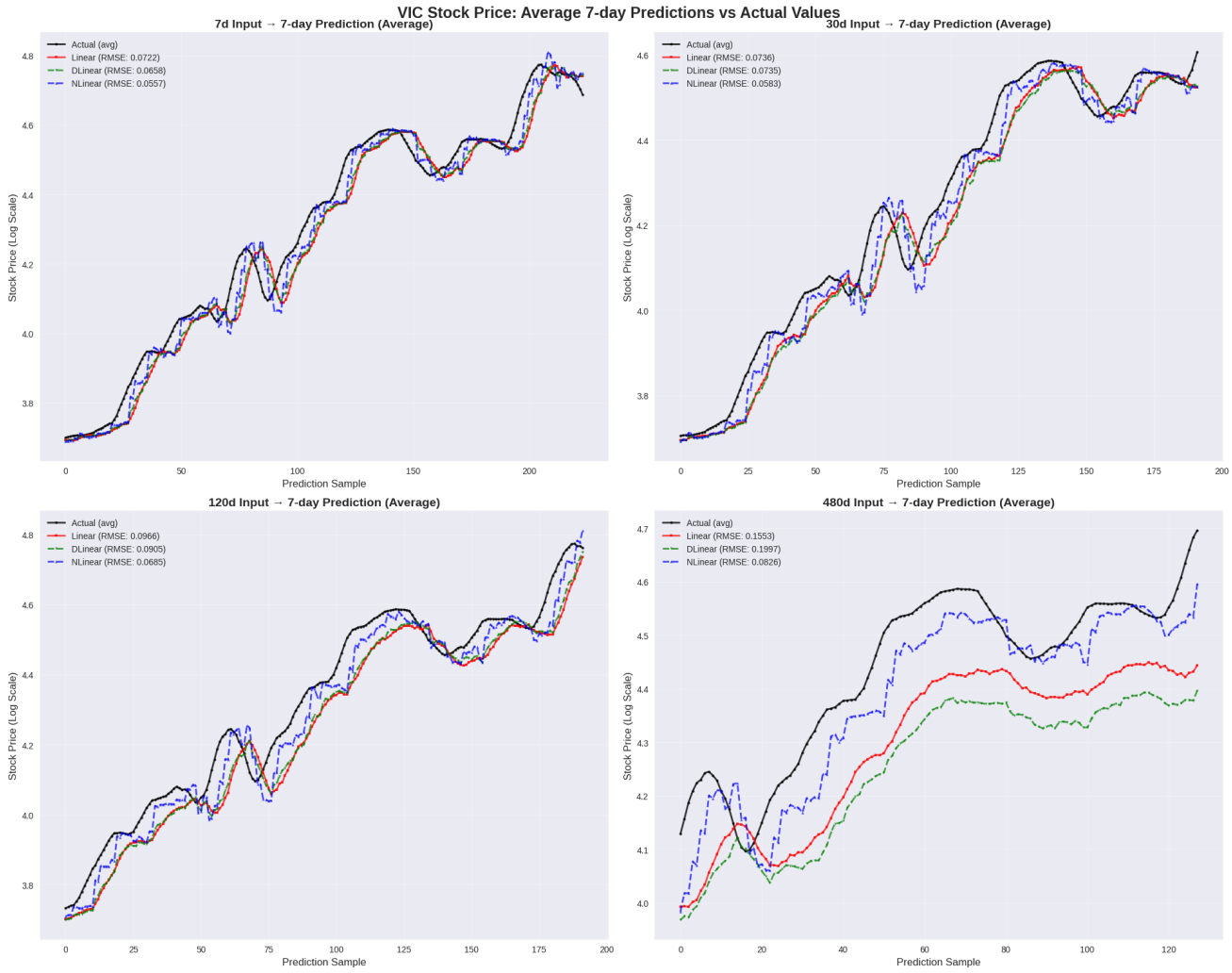
Khối lệnh trên huấn luyện toàn bộ các mô hình **Linear**, **DLinear**, **NLinear** trên bốn độ dài cửa sổ đầu vào và chỉ **vẽ** đường học (train/val) cho từng cấu hình, giúp theo dõi hội tụ mà không cần in log chi tiết.



Hình 17: Trực quan hóa loss của 4 độ dài input khác nhau

Sau khi quá trình huấn luyện hoàn tất, các mô hình Linear, DLinear và NLinear được sử dụng để dự đoán giá đóng cửa cổ phiếu VIC trong 7 ngày kế tiếp, dựa trên dữ liệu đầu vào có độ dài lần lượt là 7, 30, 120 và 480 ngày.

Các biểu đồ dưới đây trực quan hóa kết quả dự đoán, thể hiện sự so khớp giữa giá *thực tế* và *giá dự đoán* của từng mô hình. Mỗi subplot tương ứng với một độ dài cửa sổ đầu vào, giúp quan sát rõ mức độ ảnh hưởng của kích thước lịch sử đến độ chính xác của dự báo.



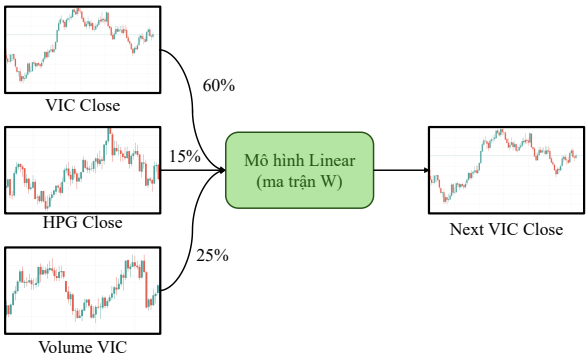
Hình 18: So sánh giữa giá thực tế và giá dự đoán từ các mô hình cho từng cửa sổ đầu vào.

VIII. Hướng cải tiến và mở rộng

Chúng ta đã thấy được sức mạnh của các mô hình LTSF-Linear. Với sự đơn giản, tốc độ và hiệu quả trong việc xử lý nhiễu thông qua các cơ chế như chuẩn hóa (NLinear) hay phân rã (DLinear), chúng đóng vai trò là một baseline cực kỳ mạnh mẽ và dễ xử dụng. Tuy nhiên, chính sự đơn giản dựa trên nền tảng tuyến tính cũng là giới hạn của lớp mô hình này. Sau khi đã có một nền tảng vững chắc, đây là hai hướng cải tiến và mở rộng tự nhiên nhất mà bạn có thể khám phá.

VIII.1. Mở rộng sang bài toán đa biến

Hạn chế hiện tại: Toàn bộ project này đang tập trung vào bài toán đơn biến, tức là chỉ dùng lịch sử giá của chính cổ phiếu VIC để dự đoán tương lai của nó. Điều này bỏ qua một thực tế quan trọng trên thị trường tài chính: giá của một cổ phiếu hiếm khi di chuyển một mình. Nó chịu ảnh hưởng mạnh mẽ từ các yếu tố bên ngoài như chỉ số thị trường chung (VN-Index), giá của các cổ phiếu đầu ngành khác (ví dụ: HPG, FPT), hoặc thậm chí là các chỉ số kinh tế vĩ mô.

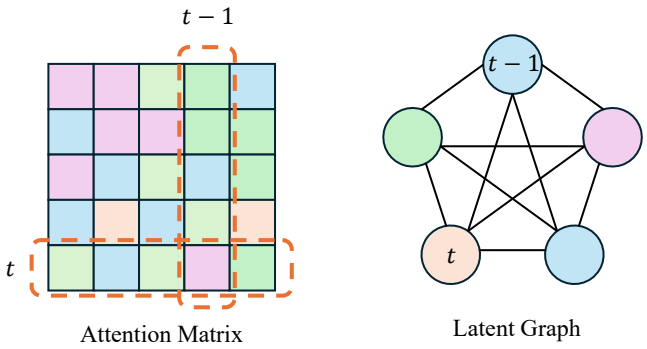


Hình 19: Với bài toán đa biến, mô hình sẽ quyết định được bao nhiêu phần trăm đóng góp mà mỗi giá trị đầu vào ảnh hưởng tới kết quả cuối cùng.

Hướng cải tiến: Hướng đi đầu tiên là **thu thập dữ liệu đa biến**, bổ sung thêm các chuỗi thời gian của các mã cổ phiếu liên quan, chỉ số VN-Index, hoặc các dữ liệu kinh tế khác. Tiếp theo, cần **điều chỉnh mô hình** bằng cách mở rộng các mô hình Linear, NLinear, và DLinear để chấp nhận đầu vào đa biến (ví dụ: input $\mathbf{x} \in \mathbb{R}^{L \times C}$ với C là số lượng biến), điều này đòi hỏi phải thay đổi kiến trúc của lớp `nn.Linear` và cách xử lý dữ liệu. Một bài toán mà chúng ta phải giải quyết là làm thế nào để áp dụng cơ chế phân rã của DLinear hay chuẩn hóa của NLinear một cách hiệu quả trên nhiều biến cùng lúc. Đây là một câu hỏi mở và là một hướng nghiên cứu thú vị.

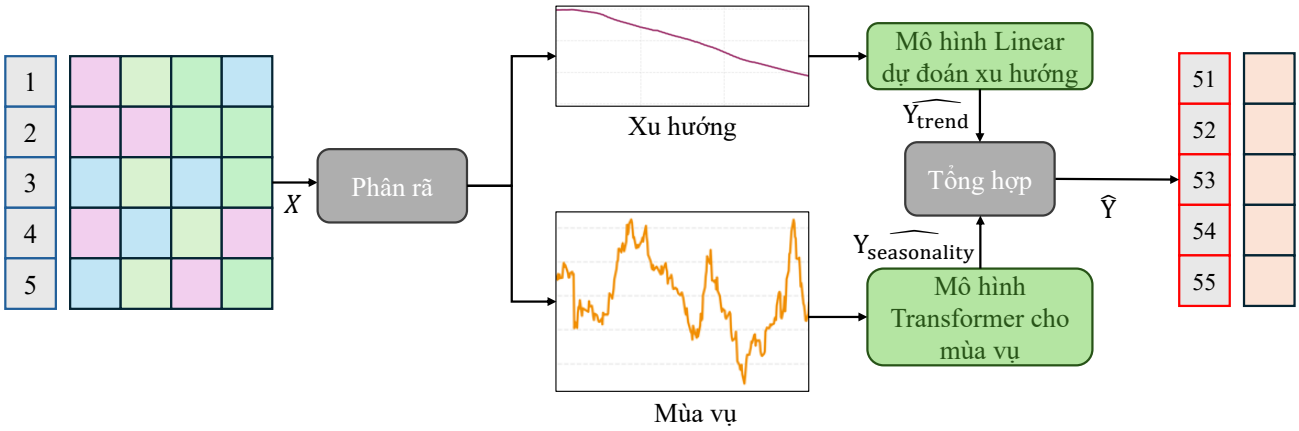
VIII.2. Hybrid model: kết hợp phân rã và transformer

Hạn chế hiện tại: Ở phần giới thiệu, chúng ta đã chỉ ra điểm yếu của Transformer là độ phức tạp tính toán và nguy cơ học vẹt nhiễu. Ngược lại, điểm mạnh nhất của DLinear là khả năng tách tín hiệu thành xu hướng và mùa vụ. Vậy tại sao không kết hợp điểm mạnh của cả hai?



Hình 20: Với lớp mô hình Transformer, khi dự đoán tương lai chúng ta có thể hình lại toàn bộ quá khứ. Đồng thời, mô hình không cần chạy qua các giai đoạn để đến được tương lai.

Hướng cải tiến: Chúng ta có thể xây dựng một mô hình lai ghép (hybrid). Bước 1 là phân rã, sử dụng moving average từ DLinear để tách chuỗi đầu vào $\mathbf{x}$ thành hai thành phần: $\mathbf{x}_{trend}$ và $\mathbf{x}_{seasonal}$. Bước 2 là xử lý hai luồng dữ liệu của chúng ta có một cách riêng biệt. Với nhánh trend ($\mathbf{x}_{trend}$), thành phần này thường rất mượt và dễ đoán nên chúng ta chỉ cần dùng một mô hình **Linear** đơn giản để dự đoán nó. Ngược lại, nhánh seasonal ($\mathbf{x}_{seasonal}$) phức tạp hơn, chứa các dao động và mẫu khó nắm bắt; đây chính là nơi một mô hình **Transformer** (ví dụ: Autoformer hoặc một Transformer đơn giản) có thể phát huy thế mạnh để tìm ra các phụ thuộc phức tạp. Cuối cùng, bước 3 là tổng hợp, cộng kết quả dự đoán từ hai nhánh lại để ra được dự đoán cuối cùng.



Hình 21: Sơ đồ huấn luyện mô hình mẫu cho cách cải tiến hybrid nói trên. Chúng ta có thể sử dùng block Attention đơn giản hoặc toàn bộ mô hình phức tạp như Autoformer vào khu vực Transformer. Cách kết hợp vẫn giữ nguyên là phép cộng.

IX. Câu hỏi trắc nghiệm

1. (Lý thuyết) Dự báo nhiều bước trực tiếp (Direct Multi-step, DMS). Phát biểu nào đúng về DMS trong các mô hình LTSF-Linear?

   (a) Dự báo tuần tự từng bước và dùng đầu ra bước t làm đầu vào bước $t{+}1$ (recursive/rollout).

   (b) Dự báo đồng thời T bước tương lai thông qua một ánh xạ duy nhất $\mathbb{R}^L \to \mathbb{R}^T$.

   (c) Cần RNN/Transformer để dự báo được T bước.

   (d) Luôn yêu cầu đổi tần suất (resampling) của chuỗi trước khi học.

2. (Lý thuyết) Trong DLinear, $\mathrm{MA}_m(\cdot)$ được dùng với mục đích chính là:

   (a) Ước lượng thành phần tần số thấp (low-pass) của chuỗi để thu được trend.

   (b) Khuếch đại thành phần tần số cao (high-pass) nhằm nhấn mạnh seasonal/residual.

   (c) Thay thế dữ liệu gốc bằng z-score.

   (d) Sinh nhãn phân lớp hai lớp.

3. (Lý thuyết) Với cửa sổ $\mathbf{x} \in \mathbb{R}^L$, chọn mốc $\ell = x_L$, đặt $\mathbf{x}' = \mathbf{x} - \ell \mathbf{1}_L$, dự báo $\hat{\mathbf{y}}' = W\mathbf{x}' + b$, hoàn nguyên $\hat{\mathbf{y}} = \hat{\mathbf{y}}' + \ell \mathbf{1}_T$. Tính chất dịch mức cộng tính (additive level-shift equivariance) nào sau đây là đúng?

   (a) $\hat{\mathbf{y}}(\mathbf{x}{+}\Delta\mathbf{1}_L) = \hat{\mathbf{y}}(\mathbf{x})$.

   (b) $\hat{\mathbf{y}}(\mathbf{x}{+}\Delta\mathbf{1}_L) = \hat{\mathbf{y}}(\mathbf{x}) + \Delta\,\mathbf{1}_T$.

   (c) $\hat{\mathbf{y}}(\mathbf{x}{+}\Delta\mathbf{1}_L) = 2\,\hat{\mathbf{y}}(\mathbf{x})$.

   (d) Không có bảo toàn nào liên quan đến dịch mức.

4. (Tính toán) MA với phép lặp biên (nghĩa lặp lại các giá trị ở biên của chuỗi dữ liệu để đệm cho các phép tính ở đầu và cuối chuỗi). Cho $m{=}3$, $\mathbf{x} = [10, 12, 11, 13]$. Tính $\mathrm{MA}_3(\mathbf{x})$ theo

$$\mathrm{MA}_3(x)_t = \tfrac{1}{3}\Big(x^*_{t-1} + x^*_t + x^*_{t+1}\Big), \quad x^* \text{ lặp biên (replicate).}$$

   (a) [10.67, 11.00, 12.00, 12.33]

   (b) [11.00, 11.00, 12.00, 12.00]

   (c) [10.00, 11.00, 11.67, 13.00]

   (d) [10.67, 11.33, 11.67, 12.33]

5. (Tính toán) NLinear (re-centering). $L{=}3$, $T{=}2$. $\mathbf{x} = [2, 3, 5]$, $\ell = x_3 = 5$. $W = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$, $b = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$. Tính $\hat{\mathbf{y}}$:

$$\mathbf{x}' = \mathbf{x} - \ell\mathbf{1}, \quad \hat{\mathbf{y}}' = W\mathbf{x}' + b, \quad \hat{\mathbf{y}} = \hat{\mathbf{y}}' + \ell\mathbf{1}.$$

   (a) [2, 2]

(b) [3, 3]

(c) [4, 4]

(d) [6, 6]

6. (Tính toán) NLinear – minh hoạ tính chất shift. Cho $L = 2, T = 1, \mathbf{x} = [2, 5]$, mốc $\ell = x_2 = 5$, $W = [1, 0]$, $b = 0$. Tính $\hat{y}(\mathbf{x})$ và $\hat{y}(\mathbf{x}+3\mathbf{1})$. Kết quả là:

   (a) $\hat{y}(\mathbf{x}) = 2,\ \hat{y}(\mathbf{x}+3\mathbf{1}) = 2$

   (b) $\hat{y}(\mathbf{x}) = 2,\ \hat{y}(\mathbf{x}+3\mathbf{1}) = 5$

   (c) $\hat{y}(\mathbf{x}) = 2,\ \hat{y}(\mathbf{x}+3\mathbf{1}) = -1$

   (d) $\hat{y}(\mathbf{x}) = 2,\ \hat{y}(\mathbf{x}+3\mathbf{1}) = 4$

7. (Tính toán) DLinear – phân rã và hai đầu tuyến tính. L=4, T=1, m=3, lặp biên. $\mathbf{x} = [10, 12, 11, 13]$, $\mathbf{x}_t = \mathrm{MA}_3(\mathbf{x})$, $\mathbf{x}_s = \mathbf{x} - \mathbf{x}_t$. $W_t = \begin{bmatrix}\frac{1}{2} & \frac{1}{2} & 0 & 0\end{bmatrix}$, $b_t = 0$; $W_s = \begin{bmatrix}1 & 0 & 0 & 0\end{bmatrix}$, $b_s = 1$. Tính $\hat{y} = (W_t\mathbf{x}_t+b_t) + (W_s\mathbf{x}_s+b_s)$.

   (a) 11.16

   (b) 12.05

   (c) 12.52

   (d) 13.09

8. (Tính toán) MSE (DMS) cho một mẫu. T=3, $\hat{\mathbf{y}} = [3, 4, 5]$, $\mathbf{y} = [2, 5, 5]$. Tính $\dfrac{1}{3}\sum_{t=1}^{3}(\hat{y}_t - y_t)^2$.

   (a) 0.2222

   (b) 0.3333

   (c) 0.6667

   (d) 1.0000

9. (Tính toán) RMSE cho 7 bước. Với $\mathbf{e} = [1, -2, 2, -1, 0, 3, -3]$, RMSE $= \sqrt{\frac{1}{7}\sum_{t=1}^{7} e_t^2}$ bằng:

   (a) 1.732

   (b) 2.000

   (c) 2.160

   (d) 2.236

10. (Tính toán) Linear – ánh xạ theo trục thời gian. L=4, T=2, $\mathbf{x} = [1, 0, 2, 1]$. $W = \begin{bmatrix}1 & 0 & 1 & 0\\ 0 & 1 & 0 & 1\end{bmatrix}$, $b = \begin{bmatrix}0\\1\end{bmatrix}$. Tính $\hat{\mathbf{y}} = W\mathbf{x} + b$.

    (a) [3, 2]

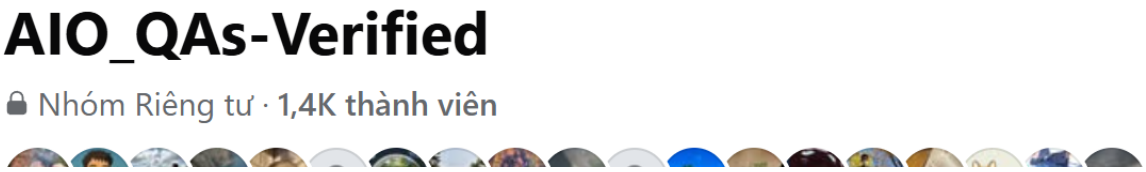
    (b) [2, 3]

    (c) [1, 4]

    (d) [4, 1]

Phụ lục

1. **Hint:** Các file code gợi ý và dữ liệu nếu có được lưu trong thư mục có thể được tải tại đây.
2. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải tại đây (Lưu ý: Sáng thứ 3 khi hết deadline phần project, ad mới copy các nội dung bài giải nêu trên vào đường dẫn).
3. **Q&A:** Bạn có thể đặt thêm câu hỏi về nội dung bài đọc trong group Facebook hỏi đáp tại đây. Tất cả câu hỏi sẽ được trả lời trong vòng tối đa 4 giờ.



Hình 22: Hình ảnh group facebook AIO Q&A

4. **Đề xuất phương pháp cải tiến project**: Một số phương pháp đề xuất cải tiến model này xem thêm ở VIII.
5. **Rubric:**

LTSF-Linear Project - Rubric		
Phần	**Kiến Thức**	**Đánh Giá**
1	- Nắm vững các khái niệm cơ bản của dự báo chuỗi thời gian. - Hiểu các đặc tính của dữ liệu tài chính (nhiễu, phi tuyến tính, không dừng) và các thành phần của chuỗi (xu hướng, mùa vụ, nhiễu).	- Phân tích tại sao một mô hình đơn giản lại có thể hoạt động tốt trong miền dữ liệu nhiễu. - Trả lời các câu hỏi trắc nghiệm lý thuyết liên quan đến khái niệm DMS và các thách thức của LTSF.
2	- Hiểu kiến trúc cơ sở của mô hình Linear. - Nắm rõ cơ chế phân rã của DLinear. - Nắm rõ cơ chế chuẩn hóa của NLinear.	- Hoàn thiện code (from scratch) cho các hàm 'forward' của ba mô hình Linear, DLinear, và NLinear. - Hoàn thiện code định nghĩa các lớp mô hình Linear, DLinear, NLinear trong project thực tế
3	- Hiểu quy trình xử lý dữ liệu chuỗi thời gian thực tế. - Nắm bắt các hướng cải tiến và mở rộng: mở rộng sang bài toán đa biến (multivariate) hoặc kết hợp mô hình lai (hybrid) như DLinear + Transformer.	- Xây dựng pipeline dữ liệu cho bài toán dự đoán giá cổ phiếu VIC. - Phân tích kết quả dự đoán và thảo luận về giới hạn cũng như các hướng cải tiến tiếp theo.