Language Models as Economic Agents

Fulin Li*

September 4, 2025

ABSTRACT

This paper shows that large language models (LLMs), built on the *transformer* architecture, can replicate the way economic agents learn from experiences. I demonstrate this through three economic problems: (1) learning transition probabilities over the business cycle, (2) predicting portfolio returns, and (3) evaluating value functions. For each problem, I provide theoretical results showing how transformers recall, weight, and incorporate past experiences into current beliefs. The transformer's parameters map to economic concepts such as the dynamic adaptive learning rates and memory recall of economic agents. Empirical evidence confirms that transformers reproduce human-like model-free learning and highlights the factors that accelerate the learning, such as adding layers or extending the context length. These findings establish a theoretical and practical foundation for training LLMs to behave like economic agents and assist with decision making.

*Texas A&M University, fli3@mays.tamu.edu. I thank Arun Akash Rangaraj for excellent research assistance.

I. Introduction

Households and firms often rely on past experiences to update their expectations and make decisions, especially when they don't have a "good" structural model of the economy. This type of *model-free learning* is well documented in behavioral macroeconomics and finance. Large language models (LLMs), built on the *transformer* architecture, operate in a simiar way: they process sequences of inputs to infer the relationships and use the learned relationships to predict outcomes. This raises a natural question: can LLMs be trained to learn and decide like economic agents?

In this paper, I show that LLMs can approach economic problems in ways that resemble economic agents with certain constraints and preferences. The paper makes three contributions. First, it opens the blackbox of LLMs by showing that, under specific parameterization, transformers replicate the adaptive, model-free learning rules of humans documented in economics and finance. In doing so, I establish a direct theoretical link between the *attention mechanism* in transformers and the way economic agents recall past experiences when forming expectations. Second, I identify the exact prompts and parameter restrictions under which transformers learn transition probabilities, predict portfolio returns, and evaluate value functions in the same way as economics agents. Finally, I provide empirical evidence that transformers not only reproduce these behaviors of economic agents but also display similar learning dynamics. Taken together, these findings offer the foundation for training LLMs to think and behave like economic agents.

I first provide a simple setup for the economic problems. There are two states of the economy — boom (H) and recession (L). Time is discrete and let s_t denote the state of the economy at time t. Assume the state transition is Markov with transition probabilities $\Pr(s_{t+1} = L | s_t = H) = \xi$ and $\Pr(s_{t+1} = H | s_t = L) = \nu$. Hence, the transition matrix is given below.

	Boom (H)	Recession (L)
Boom (H)	$1 - \xi$	ξ
Recession (L)	ν	$1 - \nu$

There is a representative agent in the economy. I assume the agent *does not know the state transition probabilities* (ξ, ν). He makes predictions about certain outcomes based on

realized states and other observables as specified below.

I present three economic problems that LLMs can approach in the same way as economic agents. To make a precise connection, I unpack the LLM blackbox and focus on the dichotomy of its backbone — the *transformer* architecture. A *transformer* is essentially an algorithm that takes a numerical matrix as input and produces another numerical matrix as output. When we use an LLM like ChatGPT, we provide a prompt — a sequence of words — that is tokenized and converted into embeddings. These embeddings form a numerical matrix representing the input sequence. The transformer underlying ChatGPT processes this matrix and generates an output matrix which is then mapped back into words.

The rest of this section provides the intuitions and main empirical findings. The formal treatment including the theoretical results is deferred to Section II.

A. Learning Transition Probabilities over the Business Cycle

The first economic problem is to learn the state transition probabilities (ξ, ν) from observed histories of the economy. Specifically, at time τ, the economic agent observes whether the economy was in a boom or a recession from time 0 through τ, and then uses this information to predict the probability of a recesssion. In this two-state setting, predicting recession odds is equivalent to learning the state transition probabilities, since the true long-run probabilities of boom and recession are given by the stationary distribution $\left(\frac{\nu}{\xi+\nu}, \frac{\xi}{\xi+\nu}\right)$.

If the economic agent does not have a structural model of the economy in mind, she can still estimate recession odds in a simple way: by computing the fraction of past periods that were recessions, i.e., the empirical probability of a recession.

Transformers can do the same. It uses a numerical vector — the *embedding* — to represent each possible state of the economy. The distance between embeddings captures the simiarity of two distinct states. This allows the transformer to retrieve and weight relevant past experiences when making predictions. Under certain parameterization, the transformer assigns higher weights to historical episodes that are "close" to recessions, effectively reproducing the empirical probability of a recession computed by the economic agent above.

B. *Predicting Portfolio Returns*

The intuition from Section I.A carries over to other prediction problems such as predicting portfolio returns in recessions. Suppose the economic agent can trade two assets: a risk-free bond fund with constant gross return R_f and an equity fund whose return depends on the state of the economy. Let $R(s_t)$ denote the gross return of the equity fund in state s_t, then $R(s_t) \in \{R_H, R_L\}$, where R_H (R_L) is the return in boom (recession). At time t, the agent invests a_t fraction of his wealth in the equity fundand the rest in the bond fund. Then the investor's gross portfolio return is $R_{p,t} = (1 - a_{t-1}) R_f + a_{t-1} R(s_t)$.

The agent is interested in learning his portfolio return at some future date $\tau + 1$ if the economy enters a recession. However, she does not know the state transition probabilities nor the set of possible return realizations $\{R_H, R_L\}$. She only observes the realized states and portfolio returns in the past.

Even without a structual model of the economy in mind, the agent can still have a "good" estimate of the portfolio return as long as at least one past episode was a recession. We can express this estimate as

$$\hat{R}_p(s_n) = \frac{1}{\tau} \sum_{t=1}^{\tau} \mathbf{1}\{s_t = s_n\} R_{p,t},$$

where $s_n = L$ corresponds to recession.

Transformers can replicate this estimate, since it retrieves memories of past recessions in the same way as described in Section I.A. More importantly, transformers (and economic agents) can go beyond this naive approach by incorporating a factor model of returns, where the factors serve as the *embeddings* of the states of the economy. When the state itself is unobservable but market returns and portfolio returns are observable, the transformer's prediction takes the form

$$\hat{R}_p(s_n) = \alpha_l + \beta_l R_M(s_n),$$

which states that the portfolio return in a recession depends on the market return, with the latter serving as a proxy for the underlying state of the economy.

The transformer updates the loadings (α_l, β_l) layer by layer. In particular, it aggregates signals that encode the time series correlation between the realized market returns and the

predicted portfolio returns from the previous layer. This process resembles an economic agent refining the mapping between economic states and portfolio returns using historical data. When the predicted portfolio return from the previous layer is below the realized portfolio return, the transformer revises its prediction upward, just as an agent adjusts her beliefs when outcomes exceed expectations.

The parameters of the transformer govern its "learning rate," which describes how "aggressively" it incorporates information from past experiences. For each historical period t, the learning rate reflects how closely other episodes align with the state realized at t. For example, if the economy was in a recession at time t, the transformer assigns higher weight to other recession periods when generating the portfolio return prediction for t — it pays more attention on data that resemble period t and adjusts the prediction accordingly.

C. Learning Value Functions

The last economic problem is to learn the lifetime utility from a given portfolio allocation rule. The setup is simiar to Barberis and Jin (2023). The agent wants to estimate her lifetime utility from a fixed portfolio allocation rule $a_t = a, \forall t = 0, 1, \ldots$. Her lifetime utility is

$$V(s_t) = \mathbb{E}\left[\sum_{\tau=t+1}^{\infty} \gamma^{t-1} \log R_{p,\tau} \middle| s_t\right],$$

where $R_{p,\tau} = (1-a) R_f + aR(s_\tau)$. Under a fixed portfolio allocation rule, his lifetime utility only depends on the current state s_t. We call $V(s_t)$ the value function, and let V^H (V^L) denote the value conditional on the economy currently being in the boom (recession).

From now on, if I invest 80% of my wealth in the stock market and 20% in the bond market. How much money will I receive after I retire? Agents do not know model premitives and use model-free learning as in behavioral finance and behavioral macro.

The agent *does not know the state transition probabililities* and he estimates V^H and V^L by observing past state and flow utility. In particular, he learns from the history of state realization $\{s_t\}_{t\in\{0,1,\ldots,\tau\}}$ and the flow utility $\{\log R_{p,t}\}_{t\in\{1,\ldots,\tau-1\}}$. We consider an agent who

implements the model-free learning as in Barberis and Jin (2023),

$$\hat{V}_{l+1}(s_n) = \hat{V}_l(s_n) + \frac{1}{n}\sum_{t=0}^{\tau-1} \lambda_l(s_t, s_n) \left(\underbrace{\log R_{p,t+1} + \gamma \hat{V}_l(s_{t+1})}_{(e)} - \underbrace{\hat{V}_l(s_t)}_{(e)'} \right). \tag{1}$$

$\hat{V}_l(s_n)$ is the agent's current estimate of the value for state s_n. She updates this estimate to $\hat{V}_{l+1}(s_n)$ according to equation (1). The term $(e)'$ above is the current estimate of the value for the state realized at time t, while the term (e) is the updated estimate incorporating the flow utility realized at time $t+1$. Hence, the difference of the two captures the prediction error. The adaptive learning rule in equation (1) says that the agent revises her estimated value for state s_n using the prediction errors over the history up to time τ. $\lambda_l(s_t, s_n)$ is the weight assigned to the prediction error at time t. Finally, the observed flow utility $\log R_{p,t}$ serves as an experience-based signal as in Ilut and Valchev (2024)

There is a transformer that implements (1) with a specific prompt and parameter restrictions. The prompt encodes three pieces of information. To provide a numerical example, consider a history of length $\tau = 3$ and suppose one period is one year. The economy starts at a boom at the beginning of year 0, which continues into year 1. Then it enters a recession at the beginning of year 2, and the recession continues into year 3. Assume the annual portfolio return has two possible values: $\log R_H = \log(1 + 15\%) = 0.14$ (in boom) and $\log R_L = \log(1 - 5\%) = 0.05$ (in recession). The embeddings for boom (H) and recession (L) are $\boldsymbol{\phi}_H, \boldsymbol{\phi}_L \in \mathbb{R}^{d\times 1}$, respectively. Then the prompt to the transformer is

$$\boldsymbol{Z}_0 = \begin{bmatrix} \boldsymbol{\phi}_H & \boldsymbol{\phi}_H & \boldsymbol{\phi}_L & \boldsymbol{\phi}_L \\ \gamma\boldsymbol{\phi}_H & \gamma\boldsymbol{\phi}_L & \gamma\boldsymbol{\phi}_L & \mathbf{0}_{d\times 1} \\ 0.14 & 0.05 & 0.05 & 0 \end{bmatrix}.$$

The first three columns of the prompt includes information on the state transition and flow utility in the past. In the last column of the prompt, $\boldsymbol{\phi}_L$ tells the transformer to estimate the value function for state L, and the zero serves as placeholders for the value function estimates.

The embeddings capture how similar one state is to another. For instance, one can represent boom and recession with numerical vectors $\boldsymbol{\phi}_H = (1, 0)^\intercal$ for boom and $\boldsymbol{\phi}_L = (0, 1)^\intercal$

for recession. Their (normalized) dot product then measures how "close" a boom is to a recession. The transformer uses this similarity measure when assigning weights to past experiences. Formally, the weight function used by the transformer is

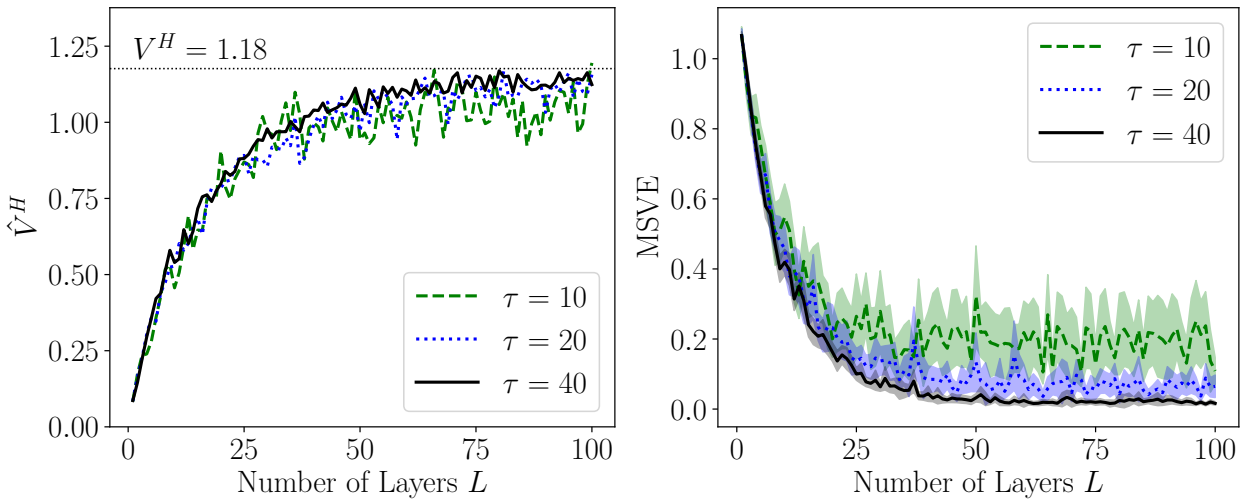
$$\lambda_l(s_t, s_n) = (\boldsymbol{\phi}(s_t))^\intercal \boldsymbol{A}_l \boldsymbol{\phi}(s_n).$$

The matrix $\boldsymbol{A}_l$ only depends on the parameters of the transformer and it determines which aspects of the past history are emphasized in the learning process. This mirrors how an economic agent recalls past experiences that are most comparable to the present.

Figure 1 reports empirical results examine on how quickly the transformer learns when trained on randomly generated policy evaluation problems. To measure the performance of the transformer, I benchmark it against an economic agent who knows all the model premitives and directly computes the value function via dynamic programming,

$$\begin{aligned} V^H &= \log R_H + \gamma\left((1-\xi)V^H + \xi V^L\right), \\ V^L &= \log R_L + \gamma\left(\nu V^H + (1-\nu)V^L\right). \end{aligned}$$

The transformer, in contrast, relies only on past state and flow utility and implements model-free learning. Figure 1 shows that the tranformer learns faster if it has more layers, while increasing the context length τ in the prompt also helps.



(a) Estimates of V^H **(b)** Estimation errors

Figure 1. Value function estimates from the transformer.

Contributions to the Literature There is a growing literaure in economics and finance that uses LLMs and the idea of embeddings to analyze the behavior of economic agents (Bybee, 2023; Horton, 2023; Fedyk et al., 2024; Gabaix et al., 2024; Manning et al., 2024; Kelly et al., 2025; Sarkar, 2025). I contribute to this literature both theoretically and empirically by opening the black box of modern LLMs. Specifically, I identify the parameter restrictions and design the prompts that instruct a transformer to behave like an economic agent. I then emprically verify that transformers can learn in the way as the theory predicted and examine the factors that affect the speed of learning. These findings bridge the gap between machine learning algorithms and economic models of decision making.

The economic problems considered in this paper include the model-free learning problems studied in the behavior economics and finance literature (Barberis and Jin, 2023; Ilut and Valchev, 2023; Ilut and Valchev, 2024). These problems are natural starting points for bridging machine learning with human learning.

The theoretical results in this paper draw insights from recent advances in reinforcement learning in computer science (Vladymyrov et al., 2024; Wang et al., 2024; Von Oswald et al., 2023). In particular, computer scientists have documented extensive evidence of *in-context learning* and *in-context reinforcement learning* in transformers — phenomena that

map naturally to how economic agents learn from past experiences.

II. LLMs as Model-Free Learners

I present a general framework for using LLMs to solve the learning problems in Section I. The framework is cast as a Markov decision problem with an arbitrary number of states. I establish the existence of transformers capable of solving such problems, and I identify both the prompt (i.e., the input data) and the parameter restrictions that enable them to "learn like humans." That is, the transformer's output coincides with economic agents' learning outcomes when given the same data.

The setup also yields natural economic interpretations. In particular, I provide theoreical results that transformers learn in an adaptive way, much like economic agents. For example, for the prediction problem in Section I.B, the transformer looks at past market returns as proxies for the state of the economy and updates its predictions about portfolio returns layer by layer. If its previous prediction is too low, it revises upward. This process mirrors how investors revise their beliefs when actual outcomes deviate from their expectations.

The parameters of a transformer govern its effective "learning rate." These parameters determine how much weight is placed on each past episode, just like how humans retrieve memories. For example, when predicting portfolio returns in future recessions, the transformer assigns higher weights to historical recessions, i.e., it uses past recessions to form expectations about returns in future recessions.

A. The Markov Decision Problem

I consider a discrete-time Markov decision problem that encompasses the learning problems introduced in Section I. The economy has a finite set of states $\mathcal{S} = \{s_1, s_2, \cdots, s_m\}$. A representative agent takes an action at each time t after the state s_t is realized. At time t, after observing the realized state $s_t \in \mathcal{S}$, the agent takes an action $a(s_t)$ from the finite set of actions $\mathcal{A}(s_t)$. The agent has period utility $u(a, s) : \mathcal{A}(s) \times \mathcal{S} \to \mathbb{R}$.

The state transition is Markov. Specifically, if state s is realized and the agent takes action $a(s)$, the probability of transitioning to state s' in the next period is given by the function $p(s, s', a(s)) : \mathcal{S} \times \mathcal{S} \times \mathcal{A}(s) \to [0, 1]$. The initial state s_0 is drawn from an initial

distribution $p_0 : \mathcal{S} \to [0, 1]$.

This setup emcompasses the learning problems introduced in Section I. For example, the problem in Section I.C can be framed as follows. The agent chooses his portfolio weights a_t according to a fixed policy function $\pi(a_t|s_t)$. His value function (i.e., lifetime utility) is given by

$$V(s_t) = u(a(s_t), s_t) + \gamma \mathbb{E}_t [V(s_{t+1})],$$

where we define the period utility (under the given policy) as $u(a(s), s) \equiv \sum_{a \in \mathcal{A}(s)} \pi(a|s) u(a, s)$, and the conditional expectation $\mathbb{E}_t[\cdot]$ is taken with respect to the state transition probability $p(s'|s) \equiv \sum_{a \in \mathcal{A}(s)} \pi(a|s) p(s'|s, a)$. The agent *does not know the state transition probabilities*. In this case, he learns the value function $V(\cdot)$ by observing the history of realized states and period utility up to time τ, $\{s_0, \log R_{p,1}, s_1, \log R_{p,2}, \ldots, s_{\tau-1}, \log R_{p,\tau}, s_\tau\}$.

B. Backbone of LLMs: Transformers and Attention Mechanism

In this section, I introduce *transformers*, an architecture that forms the backbone of most modern LLMs such as GPT and LLaMA. At a high level, a transformer is an algorithm that takes a numerical matrix as input and returns another numerical matrix as output. When we use an LLM like ChatGPT, we provide a *prompt* — a sequence of words — which is tokenized and converted into embeddings. These embeddings form a numerical matrix that represents the sequence of words. The transformer underlying ChatGPT processes this matrix, generating an output matrix which is then mapped back into words.

To evaluate whether and how LLMs learn like humans, I examine the inner workings of the transformer — specifically the *attention mechanism* — which applies a sequence of transformations to the prompt (i.e., input matrix). Importantly, I show how these transformations can be interpreted economically as learning rules documented empirically for humans, such as investors forming expectations about portfolio returns from past returns and proxies for the state of the economy. A transformer has a few key elements that naturally map to human learning: the number of layers and the parameters governing the attention mechanism. I provide theoretical results that the transformer's attention mechanism closely parallels human memory, as it retrieves and prioritizes past experiences when forming

expectations about a future outcome.

These results can be directly applied to the learning problems in Section I. In Sections II.C–II.E below, I identify the prompt and parameter restrictions under which the transformer produces an output that resembles the human learning outcomes described in Section I.

Prompt (Input) Consider a prompt $\boldsymbol{Z}_0 \in \mathbb{R}^{(D+1)\times(\tau+1)}$ with the following partition,

$$\boldsymbol{Z}_0 = \begin{bmatrix} \boldsymbol{x}_0^{(1)} & \boldsymbol{x}_0^{(2)} & \cdots & \boldsymbol{x}_0^{(\tau)} & \boldsymbol{x}_0^{(n)} \\ y_0^{(1)} & y_0^{(2)} & \cdots & y_0^{(\tau)} & 0 \end{bmatrix}, \tag{2}$$

where $\boldsymbol{x}_0^{(t)} \in \mathbb{R}^{D\times 1}$, $y_0^{(t)} \in \mathbb{R}$, $\forall t \in \{1, 2, \ldots, \tau\}$.

The first τ columns of the prompt forms the *context*, containing the relevant information for the transformer to solve a problem. We call τ the *context length*.

In the last column of the prompt, $\boldsymbol{x}_0^{(n)} \in \mathbb{R}^{D\times 1}$ is the *query*, which instructs the transformer to predict the outcome corresponding to the input $\boldsymbol{x}_0^{(n)}$. In other words, $\boldsymbol{x}_0^{(n)}$ is the "question" posed to the transformer. The bottom-right element of the prompt, $y_0^{(n)} = 0$, serves as a placeholder for the "answer" generated from the transformer. As shown below, the transformer updates its answer progressively layer by layer.

This prompt is general enough to handle all the learning problems in Section I. For example, as will be shown in Section II.D below, if the $\boldsymbol{x}_0^{(t)}$'s are realized market returns and the $y_0^{(t)}$'s are the corresponding realized portfolio returns, then the query $\boldsymbol{x}_0^{(n)}$ tells the transformer to compute the portfolio return conditional on a given market return (as a proxy for the state of the economy).

Transformer We focus on transformers with linear self-attention layers and a single attention head. Specifically, we consider a transformer with L layers, parameterized by $\{(\boldsymbol{P}_l, \boldsymbol{Q}_l)\}_{l\in\{0,1,\ldots,L-1\}}$. The transformer takes the prompt $\boldsymbol{Z}_0$ in equation (2) as input and produces, at each layer $l \in \{1, 2, \ldots, L\}$, an output matrix $\boldsymbol{Z}_l \in \mathbb{R}^{(D+1)\times(\tau+1)}$.

Attention Mechanism The transformer's output, $\boldsymbol{Z}_l$, evolves layer by layer according to

$$\boldsymbol{Z}_{l+1} = \boldsymbol{Z}_l + \frac{1}{\tau}\text{Attn}\left(\boldsymbol{Z}_l; \boldsymbol{P}_l, \boldsymbol{Q}_l\right) = \boldsymbol{Z}_l + \frac{1}{\tau}\boldsymbol{P}_l\boldsymbol{Z}_l\boldsymbol{M}\left(\boldsymbol{Z}_l^\intercal\boldsymbol{Q}_l\boldsymbol{Z}_l\right), \tag{3}$$

where the attention operator $\text{Attn}(\boldsymbol{Z}; \boldsymbol{P}, \boldsymbol{Q})$ is defined as

$$\text{Attn}(\boldsymbol{Z}; \boldsymbol{P}, \boldsymbol{Q}) \equiv \boldsymbol{P}\boldsymbol{Z}\boldsymbol{M}(\boldsymbol{Z}^\intercal \boldsymbol{Q}\boldsymbol{Z}). \tag{4}$$

The matrices $\boldsymbol{P}_l \in \mathbb{R}^{(D+1)\times(D+1)}$ and $\boldsymbol{Q}_l \in \mathbb{R}^{(D+1)\times(D+1)}$ are the parameters of the transformer at the l-th layer. The matrix $\boldsymbol{M} \in \mathbb{R}^{(\tau+1)\times(\tau+1)}$ is a fixed mask (i.e., constant across layers) on a prompt $\boldsymbol{Z}$. Throughout the paper, we focus on the case where $\boldsymbol{M}$ is diagonal, given by

$$\boldsymbol{M} = \begin{bmatrix} m_1 & 0 & \cdots & 0 & 0 \\ 0 & m_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & m_\tau & 0 \\ 0 & 0 & \cdots & 0 & m_n \end{bmatrix}. \tag{5}$$

Note that the linear transformer in equation (3) is still a nonlinear function of the input.

Output We now examine how the transformer's output is shaped by the parameters $\{(\boldsymbol{P}_l, \boldsymbol{Q}_l)\}_{l\in\{0,1,\ldots,L-1\}}$. Similar to the prompt $\boldsymbol{Z}_0$ in equation (2), we partition the output of the l-th layer, $\boldsymbol{Z}_l$, as

$$\boldsymbol{Z}_l = \begin{bmatrix} \boldsymbol{x}_l^{(1)} & \boldsymbol{x}_l^{(2)} & \cdots & \boldsymbol{x}_l^{(\tau)} & \boldsymbol{x}_l^{(n)} \\ y_l^{(1)} & y_l^{(2)} & \cdots & y_l^{(\tau)} & y_l^{(n)} \end{bmatrix}, \tag{6}$$

where $\boldsymbol{x}_l^{(i)} \in \mathbb{R}^{D\times 1}$, $y_l^{(i)} \in \mathbb{R}$, $\forall i \in \{1, 2, \ldots, \tau, n\}$.

The bottom-right element, $y_l^{(n)}$, is of particular interest since it is the "answer" to the "question" represented by $\boldsymbol{x}_l^{(n)}$. Lemma 1 establishes that it only depends on a subset of the transformer's parameters. The proof is in Appendix A.1.

Lemma 1. *Consider the L-layer transformer defined in equation* (3) *with the mask in equation* (5)*. We partition the transformer parameters* $\{(\boldsymbol{P}_l, \boldsymbol{Q}_l)\}_{l\in\{0,1,\ldots,L-1\}}$ *as*

$$\boldsymbol{P}_l = \begin{bmatrix} \boldsymbol{P}_l^a & \boldsymbol{p}_l^b \\ (\boldsymbol{p}_l^c)^\intercal & p_l^d \end{bmatrix}, \boldsymbol{Q}_l = \begin{bmatrix} \boldsymbol{Q}_l^a & \boldsymbol{q}_l^b \\ (\boldsymbol{q}_l^c)^\intercal & q_l^d \end{bmatrix},$$

where $\boldsymbol{P}_l^a,\ \boldsymbol{Q}_l^a \in \mathbb{R}^{D\times D},\ \boldsymbol{p}_l^b,\ \boldsymbol{p}_l^c,\ \boldsymbol{q}_l^b,\ \boldsymbol{q}_l^c \in \mathbb{R}^{D\times 1}$, *and* $p_l^d,\ q_l^d \in \mathbb{R}$. *At the* l*-th layer, the transformer's output has the following properties.*

(i) $\forall l \in \{1,2,\ldots,L\}$, $y_l^{(i)}$ *does not depend on* $\boldsymbol{P}_l^a$ *or* $\boldsymbol{p}_l^b$, $\forall i \in \{1,2,\ldots,\tau,n\}$.
(ii) $y_1^{(n)}$ *does not depend on* $\boldsymbol{q}_0^b$ *or* q_0^d.

From Lemma 1, without loss of generality, we focus on transformers with parameters

$$\boldsymbol{P}_l = \begin{bmatrix} \boldsymbol{0}_{D\times D} & \boldsymbol{0}_{D\times 1} \\ (\boldsymbol{p}_l^c)^\intercal & p_l^d \end{bmatrix}, \boldsymbol{Q}_l = \begin{bmatrix} \boldsymbol{Q}_l^a & \boldsymbol{q}_l^b \\ (\boldsymbol{q}_l^c)^\intercal & q_l^d \end{bmatrix}, \tag{7}$$

where $\boldsymbol{Q}_l^a \in \mathbb{R}^{D\times D}$, $\boldsymbol{p}_l^c,\ \boldsymbol{q}_l^b,\ \boldsymbol{q}_l^c \in \mathbb{R}^{D\times 1}$, and $p_l^d,\ q_l^d \in \mathbb{R}$. $\forall l \in \{1,2,\ldots,L\}$, we further define the matrix $\boldsymbol{\Sigma}_l \in \mathbb{R}^{D\times D}$, vector $\boldsymbol{\alpha}_l \in \mathbb{R}^{D\times 1}$, and scalar $\lambda_l \in \mathbb{R}$ as

$$\boldsymbol{\Sigma}_l \equiv \frac{1}{\tau}\sum_{t\in\mathcal{T}} m_t \boldsymbol{x}_l^{(t)} \left(\boldsymbol{x}_l^{(t)}\right)^\intercal, \tag{8}$$

$$\boldsymbol{\alpha}_l \equiv \frac{1}{\tau}\sum_{t\in\mathcal{T}} m_t \boldsymbol{x}_l^{(t)} y_l^{(t)}, \tag{9}$$

$$\lambda_l \equiv \frac{1}{\tau}\sum_{t\in\mathcal{T}} m_t \left(y_l^{(t)}\right)^2, \tag{10}$$

where $\mathcal{T} \equiv \{1,2,\ldots,\tau,n\}$.

Proposition 1 below characterizes the dynamics of the transformer's "answer", i.e., its output, across layers. The transformer behaves like an economic agent learning from historical data: it observes the history of "states" $\boldsymbol{x}_0^{(t)}$ and "outcomes" $y_0^{(t)}$ for $t = 1,\ldots,\tau$, and infers the underlying (nonlinear) relationship between them. Once this relationship is learned, the transformer — like an economic agent — can predict the outcome $y_l^{(n)}$ for any new query $\boldsymbol{x}_0^{(n)}$.

The learning process is progressive. At each layer l, the transformer relies on the relationship inferred from the previous layer to generate a prediction $\hat{y}_l^{(t)}$ and compares it to the realized outcome $y_0^{(t)}$. The prediction error then serves as a signal, allowing the transformer to update its beliefs about the state-outcome mapping using this signal.

Proposition 1. *Consider the L-layer transformer defined in equation* (3) *with the mask in equation* (5) *and parameters in equation* (7). *Define the vector* $\boldsymbol{c}_l \in \mathbb{R}^{D\times 1}$ *and the scalar*

$d_l \in \mathbb{R}$ *according to*

$$\begin{bmatrix} (\boldsymbol{c}_l)^\intercal & d_l \end{bmatrix} \equiv \begin{bmatrix} (\boldsymbol{p}_l^c)^\intercal & p_l^d \end{bmatrix} \begin{bmatrix} \boldsymbol{\Sigma}_l & \boldsymbol{\alpha}_l \\ \boldsymbol{\alpha}_l^\intercal & \lambda_l \end{bmatrix} \boldsymbol{Q}_l,$$

where $\boldsymbol{\Sigma}_l$, $\boldsymbol{\alpha}_l$, *and* λ_l *are defined in equations* (8)-(10). *Then* $\forall l \in \{0, 1, 2, \ldots, L-1\}$, $\forall i \in \{1, 2, \ldots, \tau, n\}$, $\boldsymbol{x}_l^{(i)} = \boldsymbol{x}_0^{(i)}$, *and there exists* $a_l \in \mathbb{R}$ *and* $\boldsymbol{w}_l \in \mathbb{R}^{D\times 1}$ *such that*

$$y_l^{(i)} = (1 + d_{l-1})\, y_{l-1}^{(i)} + \boldsymbol{c}_{l-1}^\intercal \boldsymbol{x}_0^{(i)} = a_l y_0^{(i)} - \boldsymbol{w}_l^\intercal \boldsymbol{x}_0^{(i)},$$

with $a_0 = 1$, $\boldsymbol{w}_0 = \boldsymbol{0}_{D\times 1}$, *and* a_l *and* $\boldsymbol{w}_l$ *evolves layer by layer according to*

$$\begin{aligned} a_{l+1} &= (1 + d_l)\, a_l, \\ \boldsymbol{w}_{l+1} &= (1 + d_l)\, \boldsymbol{w}_l - \boldsymbol{c}_l. \end{aligned}$$

The proof is in Appendix A.2.

Economic Interpretations of the Attention Mechanism Proposition 1 allows us to interpret the transformer's attention mechanism in terms of human memory and attention. To make this connection more transparent, I present Corollary 1, which shows the result under a specific parameterization of the transformer,

$$\boldsymbol{P}_l = \begin{bmatrix} \boldsymbol{0}_{D\times D} & \boldsymbol{0}_{D\times 1} \\ \boldsymbol{0}_{1\times D} & \eta_l \end{bmatrix}, \boldsymbol{Q}_l = \begin{bmatrix} \boldsymbol{C}_l & \boldsymbol{0}_{D\times 1} \\ \boldsymbol{0}_{1\times D} & 0 \end{bmatrix}, \tag{11}$$

where $\boldsymbol{C}_l \in \mathbb{R}^{D\times D}$ and $\eta_l \in \mathbb{R}$, $\forall l \in \{0, 1, \ldots, L-1\}$, and a mask

$$\boldsymbol{M} = \begin{bmatrix} \boldsymbol{I}_\tau & \boldsymbol{0}_{\tau\times 1} \\ \boldsymbol{0}_{1\times\tau} & 0 \end{bmatrix}. \tag{12}$$

Corollary 1. *Consider the L-layer transformer defined in equation* (3) *with the parameters in equation* (11) *and the mask in equation* (12). *Then* $\forall l \in \{0, 1, 2, \ldots, L-1\}$, $\forall t \in$

$\{1, 2, \ldots, \tau\}$, $\boldsymbol{x}_l^{(t)} = \boldsymbol{x}_0^{(t)}$, $\boldsymbol{x}_l^{(n)} = \boldsymbol{x}_0^{(n)}$, *and*

$$y_{l+1}^{(t)} = y_l^{(t)} + \frac{1}{\tau} \sum_{j=1}^{\tau} \lambda_l \left(\boldsymbol{x}_0^{(j)}, \boldsymbol{x}_0^{(t)} \right) \left(y_0^{(j)} - \boldsymbol{w}_l^\intercal \boldsymbol{x}_0^{(j)} \right), \tag{13}$$

$$y_{l+1}^{(n)} = -\boldsymbol{w}_l^\intercal \boldsymbol{x}_0^{(n)}, \tag{14}$$

where we define the function $\lambda_l(\cdot, \cdot)$ *as*

$$\lambda_l \left(\boldsymbol{x}_0^{(j)}, \boldsymbol{x}_0^{(t)} \right) \equiv \eta_l \left(\boldsymbol{x}_0^{(j)} \right)^\intercal \boldsymbol{C}_l \boldsymbol{x}_0^{(t)}, \tag{15}$$

and the vector $\boldsymbol{w}_l$ *evolves according to*

$$\boldsymbol{w}_{l+1} = \boldsymbol{w}_l - \eta_l \boldsymbol{C}_l^\intercal \boldsymbol{\alpha}_l \tag{16}$$

with initial value $\boldsymbol{w}_0 = \mathbf{0}_{D \times 1}$ *and* $\boldsymbol{\alpha}_l$ *defined in equation* (9).

The proof is in Appendix A.3.

Corollary 1 illustrates how transformers learn adaptively. We use the learning problem in Section I.B as an example, with a prompt $\boldsymbol{Z}_0$ specified as follows (using the partition in equation (2)): the $\boldsymbol{x}_0^{(t)}$'s are the realized market returns which are proxies for the state of the economy, and the $y_0^{(t)}$'s are the realized portfolio returns. At the l-th layer, the transformer predicts the portfolio return conditional the market return being $\boldsymbol{x}_0^{(n)}$, i.e., the portfolio return in a particular state of the economy, say the boom. The prediction $y_{l+1}^{(n)}$ is computed according to equation (14), using loadings $\boldsymbol{w}_l$ on the market return. Hence, the transformer answers the question: what is the portfolio return when the economy is in a boom?

The transformer updates the loadings $\boldsymbol{w}_l$ layer by layer according to equation (16). In particular, it aggregates the signals in the vector $\boldsymbol{\alpha}_l$ to make the update. As defined in equation (9), $\boldsymbol{\alpha}_l$ encodes the time series correlation between the realized market returns and the predicted portfolio returns from the previous layer. This resembles an economic agent's process of refining the mapping between the states of the economy and his portfolio returns using past data.

To generate predictions for each historical period t, the transformer follows equation (13). If the predicted portfolio return from the previous layer is below the realized portfolio return, the transformer revises its prediction upward, mirroring how an economic agent adjusts his

beliefs when outcomes exceed expectations.

The function $\lambda_l(\cdot,\cdot)$ defined in equation (15) plays the role of a "learning rate." It determines how aggressively the transformer learns from past experience over the sample period $t = 1, 2, \ldots, \tau$. For each period t in the historical data, the learning rate measures how closely other historical states align with the state realized at t. For instance, if the economy is in a boom at time t, the transformer assigns higher weight to other boom periods, paying more attention on data that resemble period t and adjusting its prediction accordingly.

For the remainder of this section, I present applications of Corollary 1 to the economic problems in Section I. In each case, I show how the transformer's output corresponds to the decision of an economic agent with specific information, constraints, and preferences, and identify the transformer parameters that give rise to this mapping. This connection also paves the way for training LLMs to "think like economic agents," as it implies a training procedure that parallels how economic agents learn from experience.

C. Application: Learning Transition Probabilities over the Business Cycle

Consider the economic problem in Section I.A, where the economic agent learns the probability of a recession based only on the realized states of the economy in the past. An agent without a structural model of the economy in mind may estimate the recession odds in a naive way: by computing the fraction of past periods that were recessions, i.e., the empirical probability of a recession.

I show that there exists a transformer that computes the recession probabilities exactly as the economic agent above. Consider the following prompt,

$$\boldsymbol{Z}_0 = \begin{bmatrix} \boldsymbol{\phi}(s_0) & \boldsymbol{\phi}(s_1) & \cdots & \boldsymbol{\phi}(s_{\tau-1}) & \boldsymbol{\phi}(s_n) \\ 1 & 1 & \cdots & 1 & 0 \end{bmatrix} \tag{17}$$

where $\forall t \in \{0, 1, \ldots, \tau - 1\}$, s_t is the state realized at time t, $\boldsymbol{\phi}(s_t) \in \mathbb{R}^{D\times 1}$ is the *embedding* of state s_t, and $\boldsymbol{\phi}(s_n)$ is the query. This prompt essentially asks the transformer to compute the outcome corresponding to the given state s_n.

The embedding is a numerical vector that represents the state of the economy. The distance between the embeddings of two different states measures their similarity, which is crucial for the transformer to identify relevant past experiences are relevant and weight them

to make predictions. Recall from Section II.A that there are m states of the economy. We can then define a matrix $\boldsymbol{\Phi} \in \mathbb{R}^{D \times m}$ that collects the embeddings of all states in $\mathcal{S}$:

$$\boldsymbol{\Phi} \equiv \begin{bmatrix} \boldsymbol{\phi}(s_1) & \boldsymbol{\phi}(s_2) & \cdots & \boldsymbol{\phi}(s_m) \end{bmatrix}. \tag{18}$$

Assumption 1. *The matrix $\boldsymbol{\Phi}$ defined in equation (18) is a permutation matrix.*

Proposition 2. *Consider a single layer transformer defined in equation (3) with $L = 1$, the mask in equation (12), and the following parameters*

$$\boldsymbol{P}_0 = \begin{bmatrix} \mathbf{0}_{D \times D} & \mathbf{0}_{D \times 1} \\ \mathbf{0}_{1 \times D} & \eta_0 \end{bmatrix}, \boldsymbol{Q}_0 = \begin{bmatrix} \boldsymbol{C}_0 & \mathbf{0}_{D \times 1} \\ \mathbf{0}_{1 \times D} & 0 \end{bmatrix}.$$

The prompt $\boldsymbol{Z}_0$ is given in equation (17). Then the transformer's output satisfies

$$y_1^{(n)} = \frac{1}{\tau} \sum_{t=0}^{\tau-1} \lambda(s_t, s_n), \tag{19}$$

where $\lambda(s_t, s_n) \equiv \eta_0 (\boldsymbol{\phi}(s_t))^\intercal \boldsymbol{C}_0 \boldsymbol{\phi}(s_n)$.

Furthermore, if the transformer's parameters satisfy $\eta_0 \boldsymbol{C}_0 = -\boldsymbol{I}_D$ and Assumption 1 holds, then the output is

$$-y_1^{(n)} = \frac{1}{\tau} \sum_{t=0}^{\tau-1} \mathbf{1}\{s_t = s_n\} \to p(s_n), \tag{20}$$

where $p(s_n)$ is the probability of state s_n in the stationary distribution.

The proof is in Appendix A.4.

Equation (20) confirms that there exists a transformer that estimates the probability of a recession exactly as the economic agent does at the beginning of this section. Equation (19) demonstrates how the transformer retrieves past experiences and weights them when computing the recession odds — it assigns higher weights to past episodes that are closer to a recession.

D. *Application: Predicting Portfolio Returns*

The insights from Proposition 2 extend to other prediction problems, including the one in Section I.B. Instead of estimating the probability of a recession, the economic agent now wants to know his portfolio return in a recession. If the economy has two states, e.g., boom and recession, then the agent can directly pin down the portfolio return as long as at least one past episode includes a recession. Put differently, he estimates the portfolio return as

$$\hat{R}_p(s_n) = \frac{1}{\tau}\sum_{t=1}^{\tau} \mathbf{1}\{s_t = s_n\} R_{p,t}, \tag{21}$$

where $s_n = L$ corresponds to recession.

In the same spirit as Proposition 2, we can show that the following prompt instructs the single layer transformer to produce the prediction in equation (21):

$$\boldsymbol{Z}_0 = \begin{bmatrix} \boldsymbol{\phi}(s_1) & \boldsymbol{\phi}(s_2) & \cdots & \boldsymbol{\phi}(s_\tau) & \boldsymbol{\phi}(s_n) \\ R_{p,1} & R_{p,2} & \cdots & R_{p,\tau} & 0 \end{bmatrix}, \tag{22}$$

provided that the transformer parameters satisfy $\eta_0 \boldsymbol{C}_0 = -\boldsymbol{I}_D$ and Assumption 1 holds. More generally, for arbitrary parameters η_0 and $\boldsymbol{C}_0$, the transformer's output is

$$y_1^{(n)} = \frac{1}{\tau}\sum_{t=1}^{\tau} \lambda(s_t, s_n) R_{p,t},$$

where the function $\lambda(\cdot,\cdot)$ assigns weights to past episodes according to their similarity to the state of interest s_n.

Although the above two-state case may appear trivial, it provides an important insight: the transformer can employ many other predictors as long as they serve as "good" proxies for the state of the economy, and in doing so it learns in the same way as economic agents who rely on those predictors. For example, suppose the market return is such a proxy. Then we can construct the following prompt:

$$\boldsymbol{Z}_0 = \begin{bmatrix} 1 & 1 & \cdots & 1 & 1 \\ R_{M,1} & R_{M,2} & \cdots & R_{M,\tau} & R_{M,\tau+1} \\ R_{p,1} & R_{p,2} & \cdots & R_{p,\tau} & 0 \end{bmatrix}, \tag{23}$$

where $R_{M,t}$ and $R_{p,t}$ denote the market return and portfolio return realized at time t, respectively. The query $\boldsymbol{x}^{(n)} = (1, R_{M,\tau+1})^\intercal$ in the last column of the prompt tells the transformer to estimate the portfolio return conditional on the market return $R_{M,\tau+1}$ being realized in the future period $\tau + 1$.

Proposition 3 establishes that the transformer can predict portfolio returns using the market return as a single factor. The result extends directly to return prediction problems with multiple factors. The proof is in Appendix A.5.

Proposition 3. *Consider the L-layer transformer defined in equation* (3) *with the mask in equation* (12) *and parameters*

$$\boldsymbol{P}_l = \begin{bmatrix} \boldsymbol{I}_D & \boldsymbol{0}_{D\times 1} \\ \boldsymbol{0}_{1\times D} & 1 \end{bmatrix}, \boldsymbol{Q}_l = \begin{bmatrix} \boldsymbol{I}_D & \boldsymbol{0}_{D\times 1} \\ \boldsymbol{0}_{1\times D} & 1 \end{bmatrix}, \tag{24}$$

$\forall l \in \{0, 1, 2, \ldots, L-1\}$. *Given the prompt in equation* (23)*, the output at the l-th layer has the following properties:*

$$-\boldsymbol{y}_l^{(n)} = \alpha_l + \beta_l \left(R_{M,\tau+1} - R_{f,\tau+1}\right), \tag{25}$$

where $\boldsymbol{w}_l \equiv (\alpha_l, \beta_l)^\intercal$ *evolves according to*

$$\begin{aligned} \boldsymbol{w}_{l+1} &= \boldsymbol{w}_l - \boldsymbol{M}_l^\intercal \boldsymbol{\alpha}_l, \\ \boldsymbol{M}_{l+1} &= \left(\boldsymbol{I}_D + \boldsymbol{\Sigma}_l\right) \boldsymbol{M}_l, \end{aligned}$$

with initial values $\boldsymbol{w}_0 = \boldsymbol{0}_{D\times 1}$, $\boldsymbol{M}_0 = \boldsymbol{I}_D$, *and* $\boldsymbol{\Sigma}_l$, $\boldsymbol{\alpha}_l$ *defined in equations* (8) *and* (9)*.*

E. Application: Learning Value Functions

Consider the economic problem in Section I.C, where the agent learns his lifetime utility from a given portfolio allocation rule based only on the realized states and portfolio returns in the past. An agent without a structural model of the economy in mind can still learn the value function adatively according to equation (1).

I show that there is a transformer that learns exactly according to equation (1) like the

economic agent. I construct the following prompt:

$$\boldsymbol{Z}_0 = \begin{bmatrix} \boldsymbol{\phi}(s_0) & \boldsymbol{\phi}(s_1) & \cdots & \boldsymbol{\phi}(s_{\tau-1}) & \boldsymbol{\phi}(s_n) \\ \gamma\boldsymbol{\phi}(s_1) & \gamma\boldsymbol{\phi}(s_2) & \cdots & \gamma\boldsymbol{\phi}(s_\tau) & \mathbf{0}_{d\times 1} \\ \log R_{p,1} & \log R_{p,2} & \cdots & \log R_{p,\tau} & 0 \end{bmatrix}, \tag{26}$$

where $\boldsymbol{\phi}(s_t) \in \mathbb{R}^{d\times 1}$ is the embedding of state s_t and thus $\boldsymbol{Z}_0 \in \mathbb{R}^{(2d+1)\times(n+1)}$. The prompt above is a special case of the general prompt in equation (2) with $D = 2d$. I impose the following restrictions on the transformer's parameters:

$$\boldsymbol{P}_l = \begin{bmatrix} \mathbf{0}_{2d\times 2d} & \mathbf{0}_{2d\times 1} \\ \mathbf{0}_{1\times 2d} & \eta_l \end{bmatrix}, \boldsymbol{Q}_l = \begin{bmatrix} -\boldsymbol{A}_l^\intercal & \boldsymbol{A}_l^\intercal & \mathbf{0}_{d\times 1} \\ \mathbf{0}_{d\times d} & \mathbf{0}_{d\times d} & \mathbf{0}_{d\times 1} \\ \mathbf{0}_{1\times d} & \mathbf{0}_{1\times d} & 0 \end{bmatrix}, \tag{27}$$

where $\eta_l \in \mathbb{R}$ and $\boldsymbol{A}_l \in \mathbb{R}^{d\times d}$.

Proposition 4 establishes that the transformer learns the value function according to (28), which coincides with the economic agent's model-free learning in (1). The proof is in Appendix A.10.

Proposition 4. *Consider the L-layer transformer defined in equation* (3) *with the mask in equation* (12) *and the parameters in equation* (27). *Let* $y_l^{(n)} \equiv \boldsymbol{Z}_l[2d+1, \tau+1]$ *denote the bottom-right element of* $\boldsymbol{Z}_l$, *where* $\boldsymbol{Z}_l$ *is the output of the l-th layer. Then* $l \in \{0, 1, \ldots, L-1\}$, *we have* $y_{l+1}^{(n)} = -V_{l+1}(s_n)$, *where* $V_l(s_n)$ *evolves according to*

$$V_{l+1}(s_n) = V_l(s_n) + \frac{1}{\tau}\sum_{t=0}^{\tau-1} \lambda_l(s_t, s_n)\left(\log R_{p,t+1} + \gamma V_l(s_{t+1}) - V_l(s_t)\right), \tag{28}$$

and the function $\lambda_l(\cdot,\cdot)$ *is defined as*

$$\lambda_l(s_t, s_n) \equiv \eta_l\left(\boldsymbol{\phi}(s_t)\right)\boldsymbol{A}_l^\intercal\boldsymbol{\phi}(s_n) \tag{29}$$

III. Conclusion

This paper demonstrates that LLMs can be understood as model-free learners whose mechanisms closely parallel economic agents' adaptive learning. By unpacking the transformer's attention mechanism, I show that state transition probabilities, portfolio returns, and value functions can all be learned from past data in the same way as economic agents. This also leads to an economic interpretation of the transformer's attention mechanism: it governs how past experiences are recalled, weighted, and incorporated into current beliefs. The empirical evidence further shows that deeper architectures and longer contexts accelerate the learning. Taken together, the findings open a path toward training LLMs to better assist with decision making.

REFERENCES

Barberis, Nicholas C, and Lawrence J Jin, 2023, Model-free and model-based learning as joint drivers of investor behavior, Technical report, National Bureau of Economic Research.

Bybee, J Leland, 2023, The ghost in the machine: Generating beliefs with large language models, *arXiv preprint arXiv:2305.02823* 349–389.

Fedyk, Anastassia, Ali Kakhbod, Peiyao Li, and Ulrike Malmendier, 2024, Chatgpt and perception biases in investments: An experimental study, *Available at SSRN* 4787249.

Gabaix, Xavier, Ralph SJ Koijen, Robert Richmond, and Motohiro Yogo, 2024, Asset embeddings, *Available at SSRN 4507511* .

Horton, John J, 2023, Large language models as simulated economic agents: What can we learn from homo silicus?, Technical report, National Bureau of Economic Research.

Ilut, Cosmin, and Rosen Valchev, 2023, Economic agents as imperfect problem solvers, *The Quarterly Journal of Economics* 138, 313–362.

Ilut, Cosmin, and Rosen Valchev, 2024, Learning optimal behavior through reasoning and experiences, *arXiv preprint arXiv:2403.18185* .

Kelly, Bryan T, Boris Kuznetsov, Semyon Malamud, and Teng Andrea Xu, 2025, Artificial intelligence asset pricing models, Technical report, National Bureau of Economic Research.

Manning, Benjamin S, Kehang Zhu, and John J Horton, 2024, Automated social science: Language models as scientist and subjects, Technical report, National Bureau of Economic Research.

Sarkar, Suproteem K, 2025, Economic representations.

Vladymyrov, Max, Johannes Von Oswald, Mark Sandler, and Rong Ge, 2024, Linear transformers are versatile in-context learners, *Advances in Neural Information Processing Systems* 37, 48784–48809.

Von Oswald, Johannes, Eyvind Niklasson, Ettore Randazzo, João Sacramento, Alexander Mordvintsev, Andrey Zhmoginov, and Max Vladymyrov, 2023, Transformers learn in-context by gradient descent, in *International Conference on Machine Learning*, 35151–35174, PMLR.

Wang, Jiuqi, Ethan Blaser, Hadi Daneshmand, and Shangtong Zhang, 2024, Transformers can learn temporal difference methods for in-context reinforcement learning, *arXiv preprint arXiv:2405.13861* .

Internet Appendix to "Language Models as Economic Agents"

A. Proofs

A.1. Proof of Lemma 1

Proof. Given the transformer in equation (3) and the partition of the l-th layer output in equation (6), $\forall i \in \{1, 2, \ldots, \tau, n\}$, we have

$$\begin{aligned}
&\begin{bmatrix} \boldsymbol{x}_{l+1}^{(i)} \\ y_{l+1}^{(i)} \end{bmatrix} - \begin{bmatrix} \boldsymbol{x}_{l}^{(i)} \\ y_{l}^{(i)} \end{bmatrix} \\
&= \frac{1}{\tau} \boldsymbol{P}_l \left(\begin{bmatrix} \boldsymbol{x}_{l}^{(n)} \\ y_{l}^{(n)} \end{bmatrix} m_n \left[\left(\boldsymbol{x}_{l}^{(n)} \right)^\intercal \quad y_{l}^{(n)} \right] + \sum_{t=1}^{\tau} \begin{bmatrix} \boldsymbol{x}_{l}^{(t)} \\ y_{l}^{(t)} \end{bmatrix} m_t \left[\left(\boldsymbol{x}_{l}^{(t)} \right)^\intercal \quad y_{l}^{(t)} \right] \right) \boldsymbol{Q}_l \begin{bmatrix} \boldsymbol{x}_{l}^{(i)} \\ y_{l}^{(i)} \end{bmatrix} \\
&= \boldsymbol{P}_l \begin{bmatrix} \boldsymbol{\Sigma}_l & \boldsymbol{\alpha}_l \\ \boldsymbol{\alpha}_l^\intercal & \lambda_l \end{bmatrix} \boldsymbol{Q}_l \begin{bmatrix} \boldsymbol{x}_{l}^{(i)} \\ y_{l}^{(i)} \end{bmatrix} \\
&= \begin{bmatrix} \boldsymbol{P}_l^a & \boldsymbol{p}_l^b \\ (\boldsymbol{p}_l^c)^\intercal & p_l^d \end{bmatrix} \begin{bmatrix} \boldsymbol{\Sigma}_l & \boldsymbol{\alpha}_l \\ \boldsymbol{\alpha}_l^\intercal & \lambda_l \end{bmatrix} \begin{bmatrix} \boldsymbol{Q}_l^a & \boldsymbol{q}_l^b \\ (\boldsymbol{q}_l^c)^\intercal & q_l^d \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_{l}^{(i)} \\ y_{l}^{(i)} \end{bmatrix},
\end{aligned} \tag{IA.1}$$

where $\boldsymbol{\Sigma}_l$, $\boldsymbol{\alpha}_l$, and λ_l are defined in equations (8)-(10). This implies

$$y_{l+1}^{(i)} = y_{l}^{(i)} + \begin{bmatrix} (\boldsymbol{p}_l^c)^\intercal & p_l^d \end{bmatrix} \begin{bmatrix} \boldsymbol{\Sigma}_l & \boldsymbol{\alpha}_l \\ \boldsymbol{\alpha}_l^\intercal & \lambda_l \end{bmatrix} \begin{bmatrix} \boldsymbol{Q}_l^a & \boldsymbol{q}_l^b \\ (\boldsymbol{q}_l^c)^\intercal & q_l^d \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_{l}^{(i)} \\ y_{l}^{(i)} \end{bmatrix},$$

which does not depend on $\boldsymbol{P}_l^a$ and $\boldsymbol{p}_l^b$. Since the prompt $\boldsymbol{Z}_0$ sets $y_0^{(n)} = 0$ (see equation (2)), the output from the first layer satisfies

$$y_{1}^{(n)} = y_{0}^{(n)} + \begin{bmatrix} (\boldsymbol{p}_0^c)^\intercal & p_0^d \end{bmatrix} \begin{bmatrix} \boldsymbol{\Sigma}_0 & \boldsymbol{\alpha}_0 \\ \boldsymbol{\alpha}_0^\intercal & \lambda_0 \end{bmatrix} \begin{bmatrix} \boldsymbol{Q}_0^a \boldsymbol{x}_0^{(n)} \\ (\boldsymbol{q}_0^c)^\intercal \boldsymbol{x}_0^{(n)} \end{bmatrix},$$

which does not depend on $\boldsymbol{q}_0^b$ and q_0^d. □

A.2. Proof of Proposition 1

Proof. Define the matrix $\boldsymbol{A}_l \in \mathbb{R}^{D\times D}$, vectors $\boldsymbol{b}_l$, $\boldsymbol{c}_l \in \mathbb{R}^{D\times 1}$, and scalar $d_l \in \mathbb{R}$ as

$$\begin{bmatrix} \boldsymbol{A}_l & \boldsymbol{b}_l \\ \boldsymbol{c}_l^\intercal & d_l \end{bmatrix} \equiv \boldsymbol{P}_l \begin{bmatrix} \boldsymbol{\Sigma}_l & \boldsymbol{\alpha}_l \\ \boldsymbol{\alpha}_l^\intercal & \lambda_l \end{bmatrix} \boldsymbol{Q}_l,$$

where $\boldsymbol{\Sigma}_l$, $\boldsymbol{\alpha}_l$, and λ_l are defined in equations (8)-(10). With the transformer parameters in equation (7), we get

$$\begin{bmatrix} \boldsymbol{A}_l & \boldsymbol{b}_l \\ \boldsymbol{c}_l^\intercal & d_l \end{bmatrix} = \begin{bmatrix} \boldsymbol{0}_{D\times D} & \boldsymbol{0}_{D\times 1} \\ (\boldsymbol{p}_l^c)^\intercal & p_l^d \end{bmatrix} \begin{bmatrix} \boldsymbol{\Sigma}_l & \boldsymbol{\alpha}_l \\ \boldsymbol{\alpha}_l^\intercal & \lambda_l \end{bmatrix} \boldsymbol{Q}_l.$$

Hence, $\boldsymbol{A}_l = \boldsymbol{0}_{D\times D}$ and $\boldsymbol{b}_l = \boldsymbol{0}_{D\times 1}$. Using equation (IA.1) in the proof of Lemma 1, we get

$$\begin{bmatrix} \boldsymbol{x}_{l+1}^{(i)} \\ y_{l+1}^{(i)} \end{bmatrix} = \begin{bmatrix} \boldsymbol{x}_l^{(i)} \\ y_l^{(i)} \end{bmatrix} + \begin{bmatrix} \boldsymbol{A}_l & \boldsymbol{b}_l \\ \boldsymbol{c}_l^\intercal & d_l \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_l^{(i)} \\ y_l^{(i)} \end{bmatrix} = \begin{bmatrix} \boldsymbol{x}_l^{(i)} \\ y_l^{(i)} \end{bmatrix} + \begin{bmatrix} \boldsymbol{0}_{D\times 1} \\ \boldsymbol{c}_l^\intercal \boldsymbol{x}_l^{(i)} + d_l y_l^{(i)} \end{bmatrix}.$$

By way of induction, $\boldsymbol{x}_{l+1}^{(i)} = \boldsymbol{x}_l^{(i)} = \boldsymbol{x}_0^{(i)}$ and thus

$$y_{l+1}^{(i)} = y_l^{(i)} + \boldsymbol{c}_l^\intercal \boldsymbol{x}_l^{(i)} + d_l y_l^{(i)} = (1 + d_l)\, y_l^{(i)} + \boldsymbol{c}_l^\intercal \boldsymbol{x}_0^{(i)}.$$

This implies $y_l^{(i)} = a_l y_0^{(i)} - \boldsymbol{w}_l^\intercal \boldsymbol{x}_0^{(i)}$ for some scalar $a_l \in \mathbb{R}$ and vector $\boldsymbol{w}_l \in \mathbb{R}^{D\times 1}$. We now derive the dynamics of a_l and $\boldsymbol{w}_l$. $\forall l \in \{0, 1, \ldots, L\}$,

$$\begin{aligned} & y_{l+1}^{(i)} = (1 + d_l)\, y_l^{(i)} + \boldsymbol{c}_l^\intercal \boldsymbol{x}_0^{(i)} \\ \Longrightarrow \quad & a_{l+1} y_0^{(i)} - \boldsymbol{w}_{l+1}^\intercal \boldsymbol{x}_0^{(i)} = (1 + d_l) \left(a_l y_0^{(i)} - \boldsymbol{w}_l^\intercal \boldsymbol{x}_0^{(i)} \right) + \boldsymbol{c}_l^\intercal \boldsymbol{x}_0^{(i)} \\ \Longrightarrow \quad & a_{l+1} = (1 + d_l)\, a_l, \boldsymbol{w}_{l+1} = (1 + d_l)\, \boldsymbol{w}_l - \boldsymbol{c}_l, \end{aligned}$$

and we set $a_0 = 1$, $\boldsymbol{w}_0 = \boldsymbol{0}_{D\times 1}$. □

A.3. Proof of Corollary 1

Proof. We first compute

$$\boldsymbol{P}_l \begin{bmatrix} \boldsymbol{\Sigma}_l & \boldsymbol{\alpha}_l \\ \boldsymbol{\alpha}_l^\intercal & \lambda_l \end{bmatrix} \boldsymbol{Q}_l = \begin{bmatrix} \boldsymbol{0}_{D\times D} & \boldsymbol{0}_{D\times 1} \\ \boldsymbol{0}_{1\times D} & \eta_l \end{bmatrix} \begin{bmatrix} \boldsymbol{\Sigma}_l & \boldsymbol{\alpha}_l \\ \boldsymbol{\alpha}_l^\intercal & \lambda_l \end{bmatrix} \begin{bmatrix} \boldsymbol{C}_l & \boldsymbol{0}_{D\times 1} \\ \boldsymbol{0}_{1\times D} & 0 \end{bmatrix} = \begin{bmatrix} \boldsymbol{0}_{D\times D} & \boldsymbol{0}_{D\times 1} \\ \eta_l \boldsymbol{\alpha}_l^\intercal \boldsymbol{C}_l & 0 \end{bmatrix},$$

where $\boldsymbol{\Sigma}_l$, $\boldsymbol{\alpha}_l$, and λ_l are defined in equations (8)-(10). Hence, $\boldsymbol{c}_l$ and d_l defined in Proposition 1 take values $\boldsymbol{c}_l = \eta_l \boldsymbol{C}_l^\intercal \boldsymbol{\alpha}_l$, $d_l = 0$. Proposition 1 implies

$$\begin{aligned} y_{l+1}^{(i)} &= y_l^{(i)} + \boldsymbol{c}_l^\intercal \boldsymbol{x}_0^{(i)} = y_l^{(i)} + \eta_l \boldsymbol{\alpha}_l^\intercal \boldsymbol{C}_l \boldsymbol{x}_0^{(i)} \\ &= y_l^{(i)} + \frac{1}{\tau}\eta_l \sum_{t\in\mathcal{T}} m_t y_l^{(t)} \left(\boldsymbol{x}_l^{(t)}\right)^\intercal \boldsymbol{C}_l \boldsymbol{x}_l^{(i)} \\ &= y_l^{(i)} + \frac{1}{\tau} \sum_{t\in\mathcal{T}} m_t y_l^{(t)} \lambda_l \left(\boldsymbol{x}_0^{(t)}, \boldsymbol{x}_0^{(i)}\right) \\ &= y_l^{(i)} + \frac{1}{\tau} \sum_{t=1}^{\tau} \lambda_l \left(\boldsymbol{x}_0^{(t)}, \boldsymbol{x}_0^{(i)}\right) \left(y_0^{(t)} - \boldsymbol{w}_l^\intercal \boldsymbol{x}_0^{(t)}\right), \end{aligned}$$

where the second line above uses the definition of $\boldsymbol{\alpha}_l$ in equation (9), and the third line above uses the definition of $\lambda_l \left(\boldsymbol{x}_0^{(t)}, \boldsymbol{x}_0^{(i)}\right)$, and the last line above uses the definition of $\boldsymbol{M}$ in equation (12). □

A.4. Proof of Proposition 2

Proof. Since we initialize $\boldsymbol{w}_0 = \boldsymbol{0}_{D\times 1}$ and $y_0^{(t)} = 1$, Corollary 1 immediately implies

$$y_1^{(n)} = y_0^{(n)} + \frac{1}{\tau} \sum_{t=1}^{\tau} \lambda \left(\boldsymbol{x}_0^{(t)}, \boldsymbol{x}_0^{(n)}\right) y_0^{(t)} = \frac{1}{\tau} \sum_{t=1}^{\tau} \lambda \left(\boldsymbol{x}_0^{(t)}, \boldsymbol{x}_0^{(n)}\right),$$

where $\lambda \left(\boldsymbol{x}_0^{(t)}, \boldsymbol{x}_0^{(n)}\right) \equiv \eta_0 \left(\boldsymbol{x}_0^{(t)}\right)^\intercal \boldsymbol{C}_0 \boldsymbol{x}_0^{(n)}$.

If the transformer parameters satisfy $\eta_0 \boldsymbol{C}_0 = \boldsymbol{I}_D$, and the embeddings are basis, then

$$\lambda\left(\boldsymbol{x}_0^{(t)}, \boldsymbol{x}_0^{(n)}\right) = \left(\boldsymbol{x}_0^{(t)}\right)^{\intercal} \boldsymbol{x}_0^{(n)} = \left(\boldsymbol{\phi}\left(s_{t-1}\right)\right)^{\intercal} \boldsymbol{\phi}\left(s_n\right) = \begin{cases} 1, & s_{t-1} = s_n \\ 0, & s_{t-1} \neq s_n \end{cases}$$

$$\Longrightarrow \quad y_1^{(n)} = \frac{1}{\tau} \sum_{t=1}^{\tau} \lambda\left(\boldsymbol{x}_0^{(t)}, \boldsymbol{x}_0^{(n)}\right) = \frac{1}{\tau} \sum_{t=1}^{\tau} \mathbf{1}\left\{s_{t-1} = s_n\right\}.$$

Using the law of large numbers for Markov chains, as $\tau \to \infty$, we have

$$y_1^{(n)} \to \mathbb{E}_p\left[\mathbf{1}\left\{s_t = s_n\right\}\right] = p\left(s_n\right).$$

□

A.5. Proof of Proposition 3

Proof. We first consider a general diagonal parameterization as follows

$$\boldsymbol{P}_l = \begin{bmatrix} p_{x,l} \boldsymbol{I}_D & \mathbf{0}_{D\times 1} \\ \mathbf{0}_{1\times D} & p_{y,l} \end{bmatrix}, \boldsymbol{Q}_l = \begin{bmatrix} q_{x,l} \boldsymbol{I}_D & \mathbf{0}_{D\times 1} \\ \mathbf{0}_{1\times D} & q_{y,l} \end{bmatrix}, \tag{IA.2}$$

where $\boldsymbol{C}_l \in \mathbb{R}^{D\times D}$ and $\eta_l \in \mathbb{R}$. Define the matrix $\boldsymbol{A}_l \in \mathbb{R}^{D\times D}$, vectors $\boldsymbol{b}_l, \boldsymbol{c}_l \in \mathbb{R}^{D\times 1}$, and scalar $d_l \in \mathbb{R}$ according to

$$\begin{bmatrix} \boldsymbol{A}_l & \boldsymbol{b}_l \\ \boldsymbol{c}_l^{\intercal} & d_l \end{bmatrix} \equiv \boldsymbol{P}_l \begin{bmatrix} \boldsymbol{\Sigma}_l & \boldsymbol{\alpha}_l \\ \boldsymbol{\alpha}_l^{\intercal} & \lambda_l \end{bmatrix} \boldsymbol{Q}_l = \begin{bmatrix} \omega_{xx,l} \boldsymbol{\Sigma}_l & \omega_{xy,l} \boldsymbol{\alpha}_l \\ \omega_{yx,l} \boldsymbol{\alpha}_l^{\intercal} & \omega_{yy,l} \lambda_l \end{bmatrix},$$

where $\boldsymbol{\Sigma}_l$, $\boldsymbol{\alpha}_l$, and λ_l are defined in equations (8)-(10), and

$$\omega_{xx,l} \equiv p_{x,l} q_{x,l}, \omega_{xy,l} \equiv p_{x,l} q_{y,l}, \omega_{yx,l} \equiv p_{y,l} q_{x,l}, \omega_{yy,l} \equiv p_{y,l} q_{y,l}. \tag{IA.3}$$

Using the transformer defined in equation (3), we get $\forall i \in \{1, 2, \ldots, \tau, n\}$,

$$\begin{bmatrix} \boldsymbol{x}_{l+1}^{(i)} \\ y_{l+1}^{(i)} \end{bmatrix} = \begin{bmatrix} \boldsymbol{x}_l^{(i)} \\ y_l^{(i)} \end{bmatrix} + \begin{bmatrix} \boldsymbol{A}_l \boldsymbol{x}_l^{(i)} + \boldsymbol{b}_l y_l^{(i)} \\ \boldsymbol{c}_l^{\intercal} \boldsymbol{x}_l^{(i)} + d_l y_l^{(i)} \end{bmatrix} = \begin{bmatrix} \boldsymbol{x}_l^{(i)} + \omega_{xx,l} \boldsymbol{\Sigma}_l \boldsymbol{x}_l^{(i)} + \omega_{xy,l} \boldsymbol{\alpha}_l y_l^{(i)} \\ y_l^{(i)} + \omega_{yx,l} \left(\boldsymbol{\alpha}_l\right)^{\intercal} \boldsymbol{x}_l^{(i)} + \omega_{yy,l} \lambda_l y_l^{(i)} \end{bmatrix}. \tag{IA.4}$$

Next, we show that there exists matrices $\boldsymbol{M}_l$, vectors $\boldsymbol{u}_l$, $\boldsymbol{w}_l$, and scalars a_l such that, $\forall t \in \{1, 2, \ldots, \tau\}$,

$$\begin{aligned} \boldsymbol{x}_l^{(t)} &= \boldsymbol{M}_l \boldsymbol{x}_0^{(t)} + y_0^{(t)} \boldsymbol{u}_l, \boldsymbol{x}_l^{(n)} = \boldsymbol{M}_l \boldsymbol{x}_0^{(n)}, \\ y_l^{(t)} &= a_l y_0^{(t)} - \boldsymbol{w}_l^\intercal \boldsymbol{x}_0^{(t)}, y_l^{(n)} = -\boldsymbol{w}_l^\intercal \boldsymbol{x}_0^{(n)}. \end{aligned} \tag{IA.5}$$

We prove the claim by induction. Since $y_0^{(n)} = 0$, we can set $a_0 = 1$, $\boldsymbol{w}_0 = \boldsymbol{0}_{D\times 1}$, $\boldsymbol{M}_0 = \boldsymbol{I}_D$, $\boldsymbol{u}_0 = \boldsymbol{0}_{D\times 1}$ at $l = 0$. Suppose (IA.5) true for some layer l, we substitute (IA.5) into the updating equation (IA.4) for layer $l+1$,

$$\begin{aligned} \begin{bmatrix} \boldsymbol{x}_{l+1}^{(t)} \\ y_{l+1}^{(t)} \end{bmatrix} &= \begin{bmatrix} \left(\boldsymbol{I}_D + \omega_{xx,l} \boldsymbol{\Sigma}_l\right) \left(\boldsymbol{M}_l \boldsymbol{x}_0^{(t)} + y_0^{(t)} \boldsymbol{u}_l\right) + \omega_{xy,l} \boldsymbol{\alpha}_l \left(a_l y_0^{(t)} - \boldsymbol{w}_l^\intercal \boldsymbol{x}_0^{(t)}\right) \\ \omega_{yx,l} \left(\boldsymbol{\alpha}_l\right)^\intercal \left(\boldsymbol{M}_l \boldsymbol{x}_0^{(t)} + y_0^{(t)} \boldsymbol{u}_l\right) + \left(1 + \omega_{yy,l} \lambda_l\right) \left(a_l y_0^{(t)} - \boldsymbol{w}_l^\intercal \boldsymbol{x}_0^{(t)}\right) \end{bmatrix}, \\ \begin{bmatrix} \boldsymbol{x}_{l+1}^{(n)} \\ y_{l+1}^{(n)} \end{bmatrix} &= \begin{bmatrix} \left(\boldsymbol{I}_D + \omega_{xx,l} \boldsymbol{\Sigma}_l\right) \boldsymbol{M}_l \boldsymbol{x}_0^{(n)} - \omega_{xy,l} \boldsymbol{\alpha}_l \boldsymbol{w}_l^\intercal \boldsymbol{x}_0^{(n)} \\ \omega_{yx,l} \left(\boldsymbol{\alpha}_l\right)^\intercal \boldsymbol{M}_l \boldsymbol{x}_0^{(n)} - \left(1 + \omega_{yy,l} \lambda_l\right) \boldsymbol{w}_l^\intercal \boldsymbol{x}_0^{(n)} \end{bmatrix}. \end{aligned}$$

Hence, we can set

$$\begin{aligned} \boldsymbol{M}_{l+1} &= \left(\boldsymbol{I}_D + \omega_{xx,l} \boldsymbol{\Sigma}_l\right) \boldsymbol{M}_l - \omega_{xy,l} \boldsymbol{\alpha}_l \boldsymbol{w}_l^\intercal \\ \boldsymbol{u}_{l+1} &= \left(\boldsymbol{I}_D + \omega_{xx,l} \boldsymbol{\Sigma}_l\right) \boldsymbol{u}_l + a_l \omega_{xy,l} \boldsymbol{\alpha}_l \\ a_{l+1} &= \left(1 + \omega_{yy,l} \lambda_l\right) a_l + \omega_{yx,l} \left(\boldsymbol{\alpha}_l\right)^\intercal \boldsymbol{u}_l \\ \boldsymbol{w}_{l+1} &= \left(1 + \omega_{yy,l} \lambda_l\right) \boldsymbol{w}_l - \omega_{yx,l} \boldsymbol{M}_l^\intercal \boldsymbol{\alpha}_l. \end{aligned}$$

With the transformer parameters in equation (IA.2), we get $\boldsymbol{u}_l = \boldsymbol{0}_{D\times 1}$, $a_l = 1$.

$$\begin{aligned} \boldsymbol{M}_{l+1} &= \left(\boldsymbol{I}_D + \boldsymbol{\Sigma}_l\right) \boldsymbol{M}_l \\ \boldsymbol{w}_{l+1} &= \boldsymbol{w}_l - \boldsymbol{M}_l^\intercal \boldsymbol{\alpha}_l. \end{aligned}$$

In particular, for a single layer transformer, the output is

$$y_1^{(n)} = -\boldsymbol{w}_1^\intercal \boldsymbol{x}_0^{(n)} = \boldsymbol{\alpha}_0^\intercal \boldsymbol{M}_0 \boldsymbol{x}_0^{(n)} = \boldsymbol{\alpha}_0^\intercal \boldsymbol{x}_0^{(n)}.$$

□

A.6. Proof of Proposition IA.1

A.7. Lemma IA.1 and Proof

Lemma IA.1. *Consider the L-layer linear transformer in equation* (3)*, which uses the mask* $\boldsymbol{M}$ *in equation* (12) *and is parameterized by* $\{(\boldsymbol{P}_l, \boldsymbol{Q}_l)\}_{l\in\{0,1,\ldots,L-1\}}$ *in equation* (**??**)*. We partition the transformer parameter* $\boldsymbol{Q}_l$ *as* $\boldsymbol{Q}_l = \begin{bmatrix} \boldsymbol{C}_l & \boldsymbol{0}_{2d\times 1} \\ \boldsymbol{0}_{1\times 2d} & 0 \end{bmatrix}$*, where* $\boldsymbol{C}_l \equiv \begin{bmatrix} -\boldsymbol{A}_l^\intercal & \boldsymbol{A}_l^\intercal \\ \boldsymbol{0}_{d\times d} & \boldsymbol{0}_{d\times d} \end{bmatrix}$.

Then the output from the l-th layer of the transformer, $\boldsymbol{Z}_l$*, has the following properties:* $\forall l \in \{1, 2, \ldots, L\}$*, we have* $\boldsymbol{X}_l = \boldsymbol{X}_0$*,* $\boldsymbol{x}_l^{(\tau+1)} = \boldsymbol{x}_0^{(\tau+1)}$*, and*

$$\boldsymbol{y}_{l+1} = \boldsymbol{y}_l + \frac{1}{\tau}\eta \boldsymbol{y}_l \boldsymbol{X}_l^\intercal \boldsymbol{C}_l \boldsymbol{X}_l, \tag{IA.6}$$

$$y_{l+1}^{(\tau+1)} = y_l^{(\tau+1)} + \frac{1}{\tau}\eta \boldsymbol{y}_l \boldsymbol{X}_l^\intercal \boldsymbol{C}_l \boldsymbol{x}_l^{(\tau+1)}, \tag{IA.7}$$

where $\boldsymbol{X}_l$*,* $\boldsymbol{x}_l^{(\tau+1)}$*,* $\boldsymbol{y}_l$*, and* $y_l^{(\tau+1)}$ *are defined according to the partition* (**??**)*.*

Proof. The proof is analogous to Lemma **??**. Using the transformer parameters $\{(\boldsymbol{P}_l, \boldsymbol{Q}_l)\}_{l\in\{0,1,\ldots,L-1\}}$ in equation (**??**) and the mask $\boldsymbol{M}$ in equation (12), we can rewrite the output from the $(l+1)$-th layer of the transformer as

$$\begin{aligned}
\boldsymbol{Z}_{l+1} &= \boldsymbol{Z}_l + \frac{1}{\tau}\boldsymbol{P}_l \boldsymbol{Z}_l \boldsymbol{M} \left(\boldsymbol{Z}_l^\intercal \boldsymbol{Q}_l \boldsymbol{Z}_l\right) \\
&= \boldsymbol{Z}_l + \frac{1}{\tau}\boldsymbol{P}_l \boldsymbol{Z}_l \boldsymbol{M} \begin{bmatrix} \boldsymbol{X}_l^\intercal & \boldsymbol{y}_l^\intercal \\ \left(\boldsymbol{x}_l^{(\tau+1)}\right)^\intercal & \left(y_l^{(\tau+1)}\right)^\intercal \end{bmatrix} \begin{bmatrix} \boldsymbol{C}_l & \boldsymbol{0}_{2d\times 1} \\ \boldsymbol{0}_{1\times 2d} & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{X}_l & \boldsymbol{x}_l^{(\tau+1)} \\ \boldsymbol{y}_l & y_l^{(\tau+1)} \end{bmatrix} \\
&= \boldsymbol{Z}_l + \frac{1}{\tau}\boldsymbol{P}_l \boldsymbol{Z}_l \begin{bmatrix} \boldsymbol{I}_\tau & \boldsymbol{0}_{\tau\times 1} \\ \boldsymbol{0}_{1\times\tau} & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{X}_l^\intercal \boldsymbol{C}_l \boldsymbol{X}_l & \boldsymbol{X}_l^\intercal \boldsymbol{C}_l \boldsymbol{x}_l^{(\tau+1)} \\ \left(\boldsymbol{x}_l^{(\tau+1)}\right)^\intercal \boldsymbol{C}_l \boldsymbol{X}_l & \left(\boldsymbol{x}_l^{(\tau+1)}\right)^\intercal \boldsymbol{C}_l \boldsymbol{x}_l^{(\tau+1)} \end{bmatrix} \\
&= \boldsymbol{Z}_l + \frac{1}{\tau}\boldsymbol{P}_l \begin{bmatrix} \boldsymbol{X}_l & \boldsymbol{x}_l^{(\tau+1)} \\ \boldsymbol{y}_l & y_l^{(\tau+1)} \end{bmatrix} \begin{bmatrix} \boldsymbol{X}_l^\intercal \boldsymbol{C}_l \boldsymbol{X}_l & \boldsymbol{X}_l^\intercal \boldsymbol{C}_l \boldsymbol{x}_l^{(\tau+1)} \\ \boldsymbol{0}_{1\times\tau} & 0 \end{bmatrix} \\
&= \boldsymbol{Z}_l + \frac{1}{\tau} \begin{bmatrix} \boldsymbol{0}_{2d\times 2d} & \boldsymbol{0}_{2d\times 1} \\ \boldsymbol{0}_{1\times 2d} & \eta \end{bmatrix} \begin{bmatrix} \boldsymbol{X}_l \boldsymbol{X}_l^\intercal \boldsymbol{C}_l \boldsymbol{X}_l & \boldsymbol{X}_l \boldsymbol{X}_l^\intercal \boldsymbol{C}_l \boldsymbol{x}_l^{(\tau+1)} \\ \boldsymbol{y}_l \boldsymbol{X}_l^\intercal \boldsymbol{C}_l \boldsymbol{X}_l & \boldsymbol{y}_l \boldsymbol{X}_l^\intercal \boldsymbol{C}_l \boldsymbol{x}_l^{(\tau+1)} \end{bmatrix} \\
&= \boldsymbol{Z}_l + \frac{1}{\tau} \begin{bmatrix} \boldsymbol{0}_{2d\times\tau} & \boldsymbol{0}_{2d\times 1} \\ \eta \boldsymbol{y}_l \boldsymbol{X}_l^\intercal \boldsymbol{C}_l \boldsymbol{X}_l & \eta \boldsymbol{y}_l \boldsymbol{X}_l^\intercal \boldsymbol{C}_l \boldsymbol{x}_l^{(\tau+1)} \end{bmatrix}.
\end{aligned}$$

Hence,

$$\begin{aligned}
&\begin{bmatrix} \boldsymbol{X}_{l+1} & \boldsymbol{x}_{l+1}^{(\tau+1)} \\ \boldsymbol{y}_{l+1} & y_{l+1}^{(\tau+1)} \end{bmatrix} = \begin{bmatrix} \boldsymbol{X}_{l} & \boldsymbol{x}_{l}^{(\tau+1)} \\ \boldsymbol{y}_{l} & y_{l}^{(\tau+1)} \end{bmatrix} + \frac{1}{\tau} \begin{bmatrix} \boldsymbol{0}_{2d\times\tau} & \boldsymbol{0}_{2d\times 1} \\ \eta \boldsymbol{y}_l \boldsymbol{X}_l^\intercal \boldsymbol{C}_l \boldsymbol{X}_l & \eta \boldsymbol{y}_l \boldsymbol{X}_l^\intercal \boldsymbol{C}_l \boldsymbol{x}_l^{(\tau+1)} \end{bmatrix} \\
\implies \quad & \boldsymbol{X}_{l+1} = \boldsymbol{X}_l, \boldsymbol{x}_{l+1}^{(\tau+1)} = \boldsymbol{x}_l^{(\tau+1)}, \\
& \boldsymbol{y}_{l+1} = \boldsymbol{y}_l + \frac{1}{\tau}\eta \boldsymbol{y}_l \boldsymbol{X}_l^\intercal \boldsymbol{C}_l \boldsymbol{X}_l, \\
& y_{l+1}^{(\tau+1)} = y_l^{(\tau+1)} + \frac{1}{\tau}\eta \boldsymbol{y}_l \boldsymbol{X}_l^\intercal \boldsymbol{C}_l \boldsymbol{x}_l^{(\tau+1)}.
\end{aligned}$$

By way of induction, $\boldsymbol{X}_l = \boldsymbol{X}_0$, $\boldsymbol{x}_l^{(\tau+1)} = \boldsymbol{x}_0^{(\tau+1)}$, $\forall l \in \{1, 2, \ldots, L\}$. □

A.8. Lemma IA.2 and Proof

Lemma IA.2. *Consider the L-layer linear transformer in equation* (3)*, which uses the mask* $\boldsymbol{M}$ *in equation* (12) *and is parameterized by* $\{(\boldsymbol{P}_l, \boldsymbol{Q}_l)\}_{l\in\{0,1,\ldots,L-1\}}$ *in equation* (**??**)*. Then*

$$y_{l+1}^{(t)} = y_0^{(t)} + \frac{1}{\tau}\eta \sum_{j=0}^{l} \boldsymbol{y}_j \boldsymbol{X}^\intercal \boldsymbol{M}_2 \boldsymbol{B}_j \boldsymbol{M}_1 \boldsymbol{x}^{(t)}, \forall l \in \{0, 1, \ldots, L-1\}, \forall t \in \{1, 2, \ldots, \tau+1\} \quad \text{(IA.8)}$$

where $\boldsymbol{X} \equiv \boldsymbol{X}_0$, $\boldsymbol{x}^{(t)} \equiv \boldsymbol{x}_0^{(t)}$, $\boldsymbol{y}_l$ *and* $y_l^{(t)}$ *are defined according to the partition* (**??**)*, and the matrices* $\boldsymbol{B}_j \equiv \begin{bmatrix} \boldsymbol{A}_j^\intercal & \boldsymbol{0}_{d\times d} \\ \boldsymbol{0}_{d\times d} & \boldsymbol{0}_{d\times d} \end{bmatrix}$, $\boldsymbol{M}_1 \equiv \begin{bmatrix} -\boldsymbol{I}_d & \boldsymbol{I}_d \\ \boldsymbol{0}_{d\times d} & \boldsymbol{0}_{d\times d} \end{bmatrix}$, $\boldsymbol{M}_2 \equiv \begin{bmatrix} \boldsymbol{I}_d & \boldsymbol{0}_{d\times d} \\ \boldsymbol{0}_{d\times d} & \boldsymbol{0}_{d\times d} \end{bmatrix}$.

Proof. The autoregressive nature of $\boldsymbol{y}_l$ and $y_l^{(\tau+1)}$ in equations (IA.6) and (IA.7) implies

$$\begin{aligned}
\boldsymbol{y}_{l+1} &= \boldsymbol{y}_0 + \frac{1}{\tau}\eta \sum_{j=0}^{l} \boldsymbol{y}_j \boldsymbol{X}^\intercal \boldsymbol{C}_j \boldsymbol{X} = \boldsymbol{y}_0 + \frac{1}{\tau}\eta \sum_{j=0}^{l} \boldsymbol{y}_j \boldsymbol{X}^\intercal \boldsymbol{M}_2 \boldsymbol{B}_j \boldsymbol{M}_1 \boldsymbol{X}, \\
y_{l+1}^{(\tau+1)} &= y_0^{(\tau+1)} + \frac{1}{\tau}\eta \sum_{j=0}^{l} \boldsymbol{y}_j \boldsymbol{X}^\intercal \boldsymbol{C}_j \boldsymbol{x}^{(\tau+1)} = y_0^{(\tau+1)} + \frac{1}{\tau}\eta \sum_{j=0}^{l} \boldsymbol{y}_j \boldsymbol{X}^\intercal \boldsymbol{M}_2 \boldsymbol{B}_j \boldsymbol{M}_1 \boldsymbol{x}^{(\tau+1)},
\end{aligned}$$

where we use the fact that $\boldsymbol{M}_2 \boldsymbol{B}_j = \boldsymbol{B}_j$ by construction and $\boldsymbol{C}_j$ defined in Lemma IA.1 satisfies $\boldsymbol{C}_j = \boldsymbol{B}_j \boldsymbol{M}_1$. □

A.9. Lemma IA.3 and Proof

Lemma IA.3. *Define* $\boldsymbol{\psi}_0 \equiv \mathbf{0}_{2d\times 1}$ *and* $\boldsymbol{\psi}_l \equiv \frac{1}{\tau}\eta \sum_{j=0}^{l-1} \boldsymbol{B}_j^\intercal \boldsymbol{M}_2 \boldsymbol{X}\boldsymbol{y}_j^\intercal$ *for* $l \in \{1, 2, \ldots, L\}$ *with* $\boldsymbol{B}_j$ *and* $\boldsymbol{M}_2$ *defined in Lemma IA.2. Then there exists a sequence* $\{\boldsymbol{w}_l\}_{l\in\{0,1,\ldots,L\}}$ *such that* $\boldsymbol{\psi}_l = \begin{bmatrix} \boldsymbol{w}_l \\ \mathbf{0}_{d\times 1} \end{bmatrix}$.

Proof. $\forall l \in \{1, 2, \ldots, L\}$,

$$\boldsymbol{\psi}_l \equiv \frac{1}{\tau}\eta \sum_{j=0}^{l-1} \boldsymbol{B}_j^\intercal \boldsymbol{M}_2 \boldsymbol{X}\boldsymbol{y}_j^\intercal = \frac{1}{\tau}\eta \sum_{j=0}^{l-1} \boldsymbol{B}_j^\intercal \boldsymbol{X}\boldsymbol{y}_j^\intercal = \frac{1}{\tau}\eta \sum_{j=0}^{l-1} \begin{bmatrix} \boldsymbol{A}_j & \mathbf{0}_{d\times d} \\ \mathbf{0}_{d\times d} & \mathbf{0}_{d\times d} \end{bmatrix} \boldsymbol{X}\boldsymbol{y}_j^\intercal,$$

where the second equality uses the fact that $\boldsymbol{B}_j^\intercal \boldsymbol{M}_2 = \boldsymbol{B}_j^\intercal$. □

A.10. Proof of Proposition 4

I provide proposition IA.1 below which immediately implies the result in Proposition 4.

Proposition IA.1. *Consider the L-layer linear transformer in equation* (3)*, which uses the mask* $\boldsymbol{M}$ *in equation* (12) *and is parameterized by* $\{(\boldsymbol{P}_l, \boldsymbol{Q}_l)\}_{l\in\{0,1,\ldots,L-1\}}$ *in equation* (**??**)*. Let* $y_l^{(\tau+1)} \equiv \boldsymbol{Z}_l[2d+1, \tau+1]$ *denote the bottom-right element of* $\boldsymbol{Z}_l$*, where* $\boldsymbol{Z}_l$ *is the output from the l-th layer of the transformer,* $l \in \{1, 2, \ldots, L\}$*. Then* $y_l^{(\tau+1)} = -\boldsymbol{w}_l^\intercal \boldsymbol{\phi}(s_\tau)$*, where* $\boldsymbol{w}_0 = \mathbf{0}_{d\times 1}$ *and*

$$\boldsymbol{w}_{l+1} = \boldsymbol{w}_l + \frac{1}{\tau}\eta \boldsymbol{A}_l \sum_{t=0}^{\tau-1} \left(\log R_{p,t+1} + \gamma \boldsymbol{w}_l^\intercal \boldsymbol{\phi}(s_{t+1}) - \boldsymbol{w}_l^\intercal \boldsymbol{\phi}(s_t)\right) \boldsymbol{\phi}(s_t). \tag{IA.9}$$

Proof. Lemma IA.2 and IA.3 imply

$$\begin{aligned} y_l^{(\tau+1)} &= y_0^{(\tau+1)} + \frac{1}{\tau}\eta \sum_{j=0}^{l-1} \boldsymbol{y}_j \boldsymbol{X}^\intercal \boldsymbol{M}_2 \boldsymbol{B}_j \boldsymbol{M}_1 \boldsymbol{x}^{(\tau+1)} \\ &= \boldsymbol{\psi}_l^\intercal \boldsymbol{M}_1 \boldsymbol{x}^{(\tau+1)} = \begin{bmatrix} \boldsymbol{w}_l^\intercal & \mathbf{0}_{1\times d} \end{bmatrix} \begin{bmatrix} -\boldsymbol{I}_d & \boldsymbol{I}_d \\ \mathbf{0}_{d\times d} & \mathbf{0}_{d\times d} \end{bmatrix} \begin{bmatrix} \boldsymbol{\phi}(s_\tau) \\ \mathbf{0}_{d\times 1} \end{bmatrix} \\ &= -\boldsymbol{w}_l^\intercal \boldsymbol{\phi}(s_\tau). \end{aligned}$$

The dynamics of $y_l^{(\tau+1)}$ from equation (IA.7) of Lemma IA.1 implies the dynamics of $\{\boldsymbol{w}_l\}_{l\in\{0,\ldots,L\}}$ from one layer to the next. We first compute

$$
\begin{aligned}
& \frac{1}{\tau}\eta \boldsymbol{y}_l \boldsymbol{X}^\intercal \boldsymbol{C}_l \boldsymbol{x}^{(\tau+1)} \\
= \; & \frac{1}{\tau}\eta \boldsymbol{y}_l \boldsymbol{X}^\intercal \begin{bmatrix} -\boldsymbol{A}_l^\intercal & \boldsymbol{A}_l^\intercal \\ \boldsymbol{0}_{d\times d} & \boldsymbol{0}_{d\times d} \end{bmatrix} \begin{bmatrix} \boldsymbol{\phi}(s_\tau) \\ \boldsymbol{0}_{d\times 1} \end{bmatrix} \\
= \; & \frac{1}{\tau}\eta \boldsymbol{y}_l \begin{bmatrix} \boldsymbol{\phi}(s_0)^\intercal & \gamma(\boldsymbol{\phi}(s_1))^\intercal \\ \boldsymbol{\phi}(s_1)^\intercal & \gamma(\boldsymbol{\phi}(s_2))^\intercal \\ \vdots & \vdots \\ \boldsymbol{\phi}(s_{\tau-1})^\intercal & \gamma(\boldsymbol{\phi}(s_\tau))^\intercal \end{bmatrix} \begin{bmatrix} -\boldsymbol{A}_l^\intercal \boldsymbol{\phi}(s_\tau) \\ \boldsymbol{0}_{d\times 1} \end{bmatrix} \\
= \; & -\frac{1}{\tau}\eta \begin{bmatrix} y_l^{(1)} & y_l^{(2)} & \cdots & y_l^{(n)} \end{bmatrix} \begin{bmatrix} \boldsymbol{\phi}(s_0)^\intercal \boldsymbol{A}_l^\intercal \boldsymbol{\phi}(s_\tau) \\ \boldsymbol{\phi}(s_1)^\intercal \boldsymbol{A}_l^\intercal \boldsymbol{\phi}(s_\tau) \\ \vdots \\ \boldsymbol{\phi}(s_{\tau-1})^\intercal \boldsymbol{A}_l^\intercal \boldsymbol{\phi}(s_\tau) \end{bmatrix} \\
= \; & -\frac{1}{\tau}\eta \sum_{t=0}^{\tau-1} \left(\log R_{p,t+1} + \boldsymbol{\psi}_l^\intercal \boldsymbol{M}_1 \boldsymbol{x}^{(t+1)}\right) (\boldsymbol{\phi}(s_t))^\intercal \boldsymbol{A}_l^\intercal \boldsymbol{\phi}(s_\tau) \\
= \; & -\frac{1}{\tau}\eta \sum_{t=0}^{\tau-1} \left(\log R_{p,t+1} + \begin{bmatrix} -\boldsymbol{w}_l \\ \boldsymbol{w}_l \end{bmatrix}^\intercal \begin{bmatrix} \boldsymbol{\phi}(s_t) \\ \gamma\boldsymbol{\phi}(s_{t+1}) \end{bmatrix}\right) (\boldsymbol{\phi}(s_t))^\intercal \boldsymbol{A}_l^\intercal \boldsymbol{\phi}(s_\tau) \\
= \; & -\frac{1}{\tau}\eta \sum_{t=0}^{\tau-1} \left(\log R_{p,t+1} + \gamma \boldsymbol{w}_l^\intercal \boldsymbol{\phi}(s_{t+1}) - \boldsymbol{w}_l^\intercal \boldsymbol{\phi}(s_t)\right) (\boldsymbol{\phi}(s_t))^\intercal \boldsymbol{A}_l^\intercal \boldsymbol{\phi}(s_\tau),
\end{aligned}
$$

where the second line above uses the definition of $\boldsymbol{C}_l$ in Lemma IA.1, the third and fourth lines above use the definition of $\boldsymbol{X}_l$ and $y_l^{(t)}$ in partition (**??**), the fifth line above uses Lemma IA.2, and the sixth line above uses Lemma IA.3. Hence, the dynamics of $y_l^{(\tau+1)}$ from equation (IA.7) of Lemma IA.1 implies

$$
-\boldsymbol{w}_{l+1}^\intercal \boldsymbol{\phi}(s_\tau) = -\boldsymbol{w}_l^\intercal \boldsymbol{\phi}(s_\tau) - \frac{1}{\tau}\eta \sum_{t=0}^{\tau-1} \left(\log R_{p,t+1} + \gamma \boldsymbol{w}_l^\intercal \boldsymbol{\phi}(s_{t+1}) - \boldsymbol{w}_l^\intercal \boldsymbol{\phi}(s_t)\right) (\boldsymbol{\phi}(s_t))^\intercal \boldsymbol{A}_l^\intercal \boldsymbol{\phi}(s_\tau),
$$

Since above dynamics holds for any $\boldsymbol{\phi}(s_\tau)$, we have

$$\boldsymbol{w}_{l+1} = \boldsymbol{w}_l + \frac{1}{\tau}\eta \boldsymbol{A}_l \sum_{t=0}^{\tau-1} \left(\log R_{p,t+1} + \gamma \boldsymbol{w}_l^\intercal \boldsymbol{\phi}(s_{t+1}) - \boldsymbol{w}_l^\intercal \boldsymbol{\phi}(s_t)\right) \boldsymbol{\phi}(s_t).$$

□