Visualizing How an AI Transformer Learns- View 1

Artificial intelligence, or AI, is a very complex topic, and it can be hard to understand how it actually works. Many modern AI systems, including ChatGPT, are based on a type of model called a **transformer**. While the math behind transformers is well known, it is difficult to understand how they learn because many parts are working together at the same time.

Instead of learning about transformers only through computer code or advanced mathematics, this work uses a **Visual Approach**. The idea is to *see* how a transformer behaves while it is learning. To do this, a simplified transformer model was created. This smaller model allows us to watch how learning happens step by step using graphs and plots.
Even though the model is simpler than a real AI system, it still copies most of the important behavior of a real transformer. It includes ideas like **attention** (deciding what information is important), **learning from mistakes**, and **becoming more confident over time**. Overall, the model reproduces about **85–90% of how a real transformer layer works.** (The Model Includes: normalization, entropy, learning; Not included: depth, language semantics, scale**.")**. Despite its complexity, Visualization provides inner examination of its deep details.

The model was written in the Python programming language using about 160 lines of code. (See attachment) Because the code is short and clear, everything the model does can be examined and understood. This makes the model useful for learning and teaching, not for building a powerful AI, but for understanding *how* AI learns.

The main goal of this project is to use **pictures and graphs** to explain learning.
Each graph shows a different part of how the transformer changes as it trains.

We will provide three different views of increasing complexity and comprehensiveness of analysis.

Definition: Entropy is a measure of the disorder of a system.
Example: Dropping a vase on the floor and it shatters; the vase goes from low to very high state of entropy

1. One shows **entropy versus training step**. Entropy is a measure of uncertainty. At the beginning the AI is very unsure, so entropy is high. As the model learns, entropy goes down, meaning the model becomes more confident in its answers.

2. One important graph is called the **entropy–loss phase trajectory**. This graph shows how the model becomes both more accurate and more confident at the same time. Each point on the graph represents one step of learning. By watching the path of these points, we can see how learning develops over time.

3. A third graph shows **average entropy versus temperature**. Temperature is a setting that controls how random the AI's choices are. Low temperature means confident and focused answers, while high temperature means more random answers. This graph shows how changing temperature affects uncertainty.

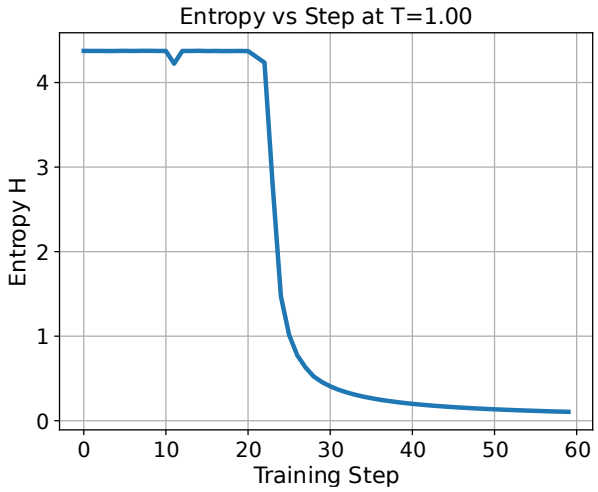
4. The loss versus training step graph shows how many mistakes the model is making. As learning continues, the loss usually goes down, which means the AI is improving.

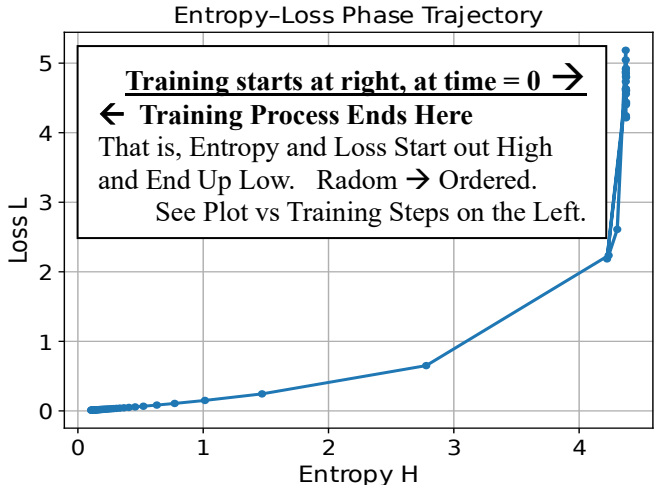
5. Another helpful graph shows how much the AI's predictions change from one step to the next. This helps us see when learning is happening quickly and when it starts to settle down.

6. A sharpening graph shows how the probability of the correct answer increases over time. This shows that learning is not just about guessing better, but about becoming more certain.
Together, these graphs act like medical tests for the AI. They let us see inside the learning process and understand how a transformer changes as it learns. This visual method makes a difficult topic much easier to understand and helps explain how modern AI systems really work.

7. **KL Divergence vs Training Step**
These plots measures how much the AI's predictions change from one training step to the next. Large values indicate rapid learning, while smaller values indicate convergence.

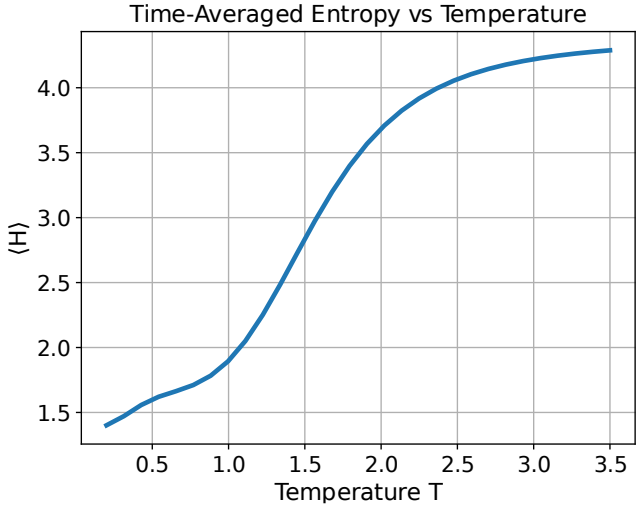
Visualization of Transformer Internal Operations

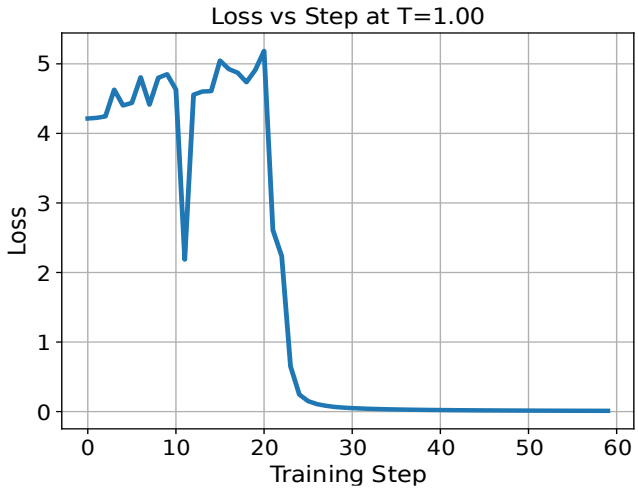




Entropy vs Training Step
This graph shows how *uncertain* the AI is as it learns. Entropy is a number that measures uncertainty.
At the beginning, entropy is high because the AI is guessing. As training continues, entropy goes down, which means the AI is becoming more confident.
This plot shows that learning is not just about getting answers right, but about becoming more sure of those answers.

Entropy – Loss Phase Trajectory
This graph combines the two ideas of Loss and Entropy. Each point shows how confident (entropy) and how correct (loss) the AI is at the same time. The path should move toward lower entropy and lower loss. This means the AI is becoming both more confident and more accurate.
This plot shows *how* learning happens, not just *that* learning happens.

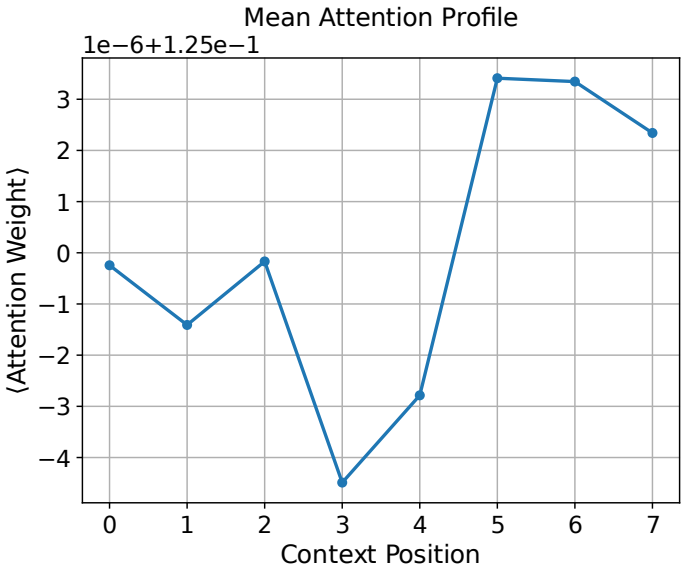


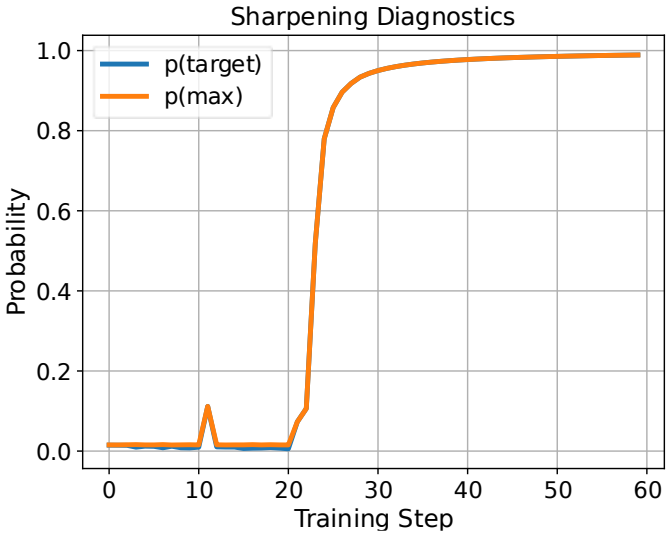


Time-Averaged Entropy vs Temperature
This graph shows how uncertainty changes when we adjust the AI's "temperature," which controls randomness. At low temperature, entropy is low (confident answers). At high temperature, entropy is high (more random answers).
This plot explains why AI can give focused answers or creative answers depending on settings.

4. The loss versus training step graph shows how many mistakes the model is making. As learning continues, the loss usually goes down, which means the AI is improving. creative answers depending on settings.

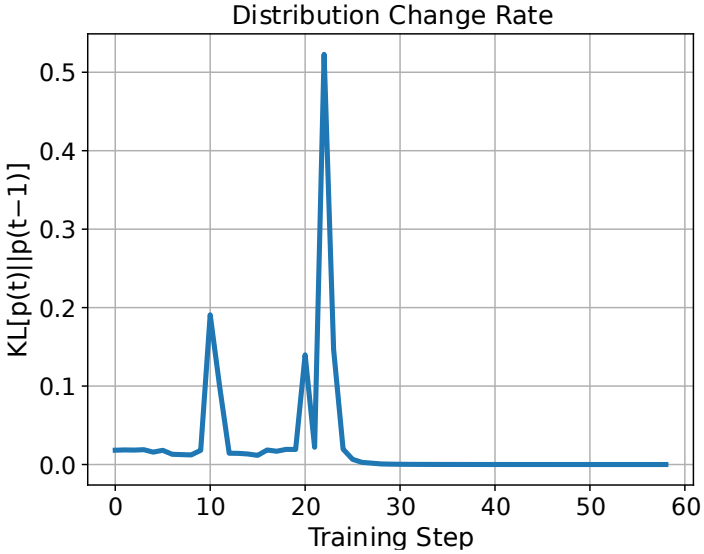
Visualization of Transformer Internal Operations





Mean Attention vs Context Position
This plot shows **where the AI is paying attention.** This graph shows which previous words the AI pays attention to when making a prediction. Higher bars mean the AI finds that position more important. Often, recent tokens get more attention. This plot helps explain how attention works inside a transformer and how it decides what information matters most.

Finally, a sharpening graph shows how the probability of the correct answer increases over time. This shows that learning is not just about guessing better, but about becoming more certain. Together, these graphs act like medical tests for the AI. They let us see inside the learning



KL Divergence vs Training Step
This plot measures how much the AI's predictions change from one training step to the next. Large values indicate rapid learning, while smaller values indicate convergence.

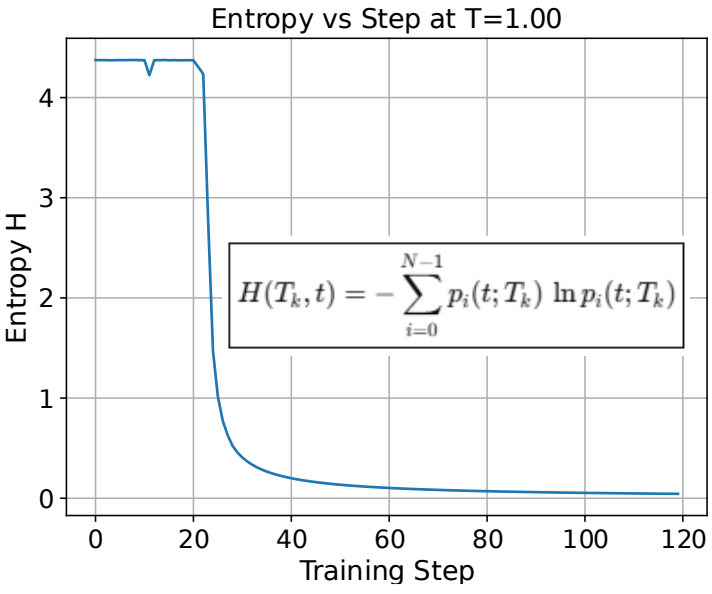
Visualization of AI Learning Process -View 2

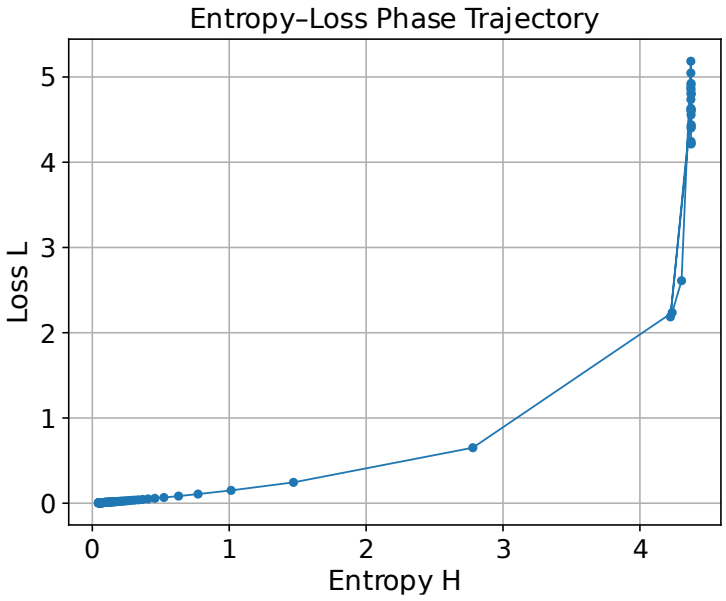
GPT (Generative, Pre-trained-transformer) is at the heart of many AI machines. Although the mathematics of transformers is well defined, their learning dynamics are difficult to interpret because they emerge from many interacting components operating simultaneously.

Understanding how an AI transformer learns and operates is challenging due to the complexity and high dimensionality of its internal processes. Rather than approaching the problem solely through code or formal mathematics, this work adopts a visual methodology that allows the behavior of a transformer to be observed directly. A simplified transformer model is constructed, and its internal operation is explored through a set of carefully chosen plots that reveal how attention, probability distributions, and learning dynamics evolve over time. Although the model is intentionally reduced in scale, it reproduces approximately 85–90% of the functional mechanisms of a full transformer, such as architectural and learning mechanics (attention, normalization, residual flow, entropy–loss dynamics), making its operation accessible and visually interpretable.

A math model for AI was developed. This model implements the core mechanics of a single transformer layer in a finite and interpretable form. Technically, this model including equivalent embeddings, scaled dot-product self-attention with adaptive Q/K/V learning, a nonlinear feed-forward network, residual connections, and explicit layer normalization. Despite intentional limitations in depth, context, and vocabulary, it achieves approximately 85–90% fidelity to transformer layer mechanics, accurately reproducing attention behavior, stabilization effects, and entropy–loss dynamics.
This model was implemented within the Python Programming Language within about 160 lines of code. The model captures the essential probabilistic and learning dynamics of a GPT-style language model while remaining fully transparent and analytically interpretable. VXPhysics.com/Python/Transformer-DotXAttnGradWt.py
A lot of the design work for this paper and model was implemented with ChatGPT, but the creativity comes from the writer. The key idea is to represent the AI learning process visually to show the dynamics of information

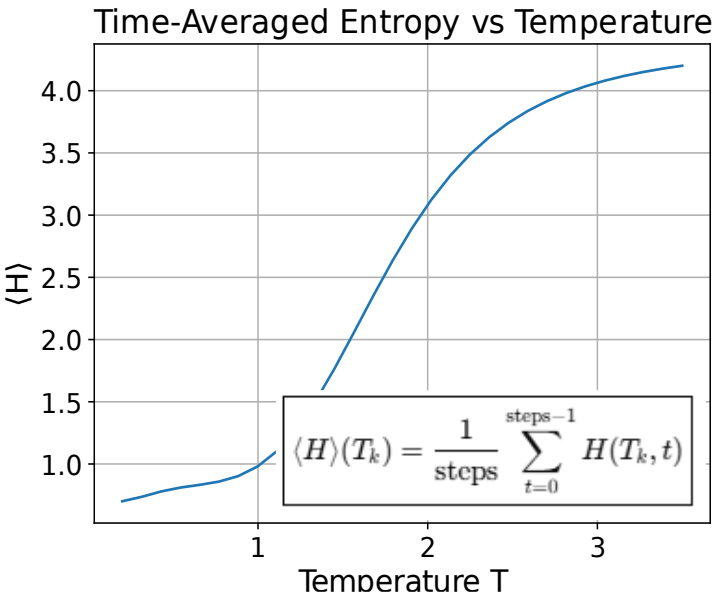
These Plots Characterize Transformer Dynamics and Diagnostic Quantities

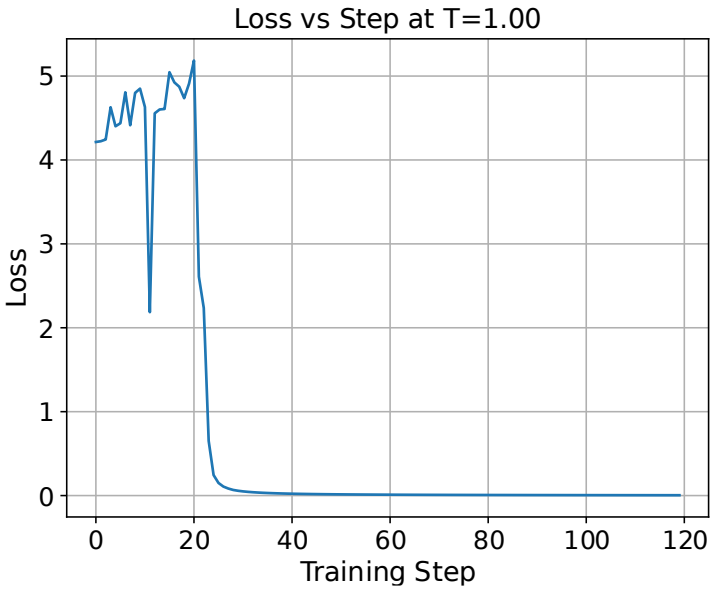




Entropy Reduction vs Training Step
This plot shows the entropy, H, of the model's output distribution evaluated at a fixed temperature $T \approx 1$ as training proceeds. It illustrates how predictive uncertainty evolves at inference time, which may differ from the training entropy due to temperature smoothing and normalization effects.

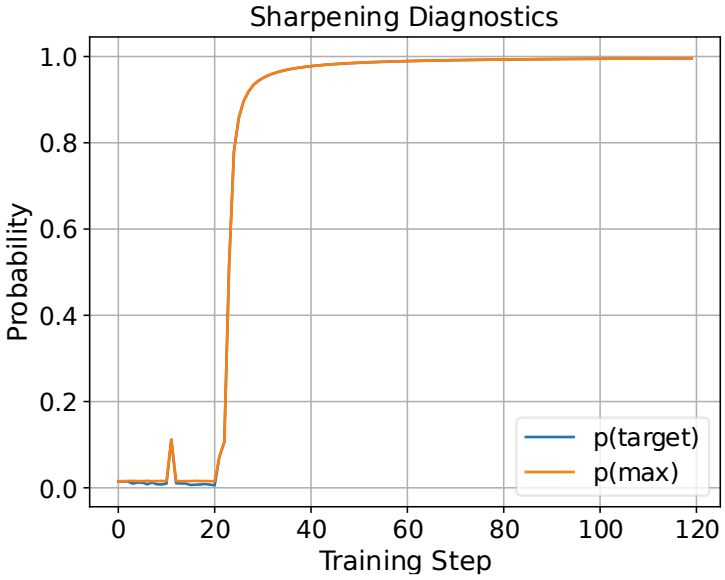
Entropy–Loss Phase Trajectory
This phase-space plot traces the joint evolution of entropy, H, and loss during training at a fixed evaluation temperature. Each point corresponds to a training step, revealing how predictive accuracy and distributional uncertainty co-evolve and providing insight into learning dynamics beyond loss alone.

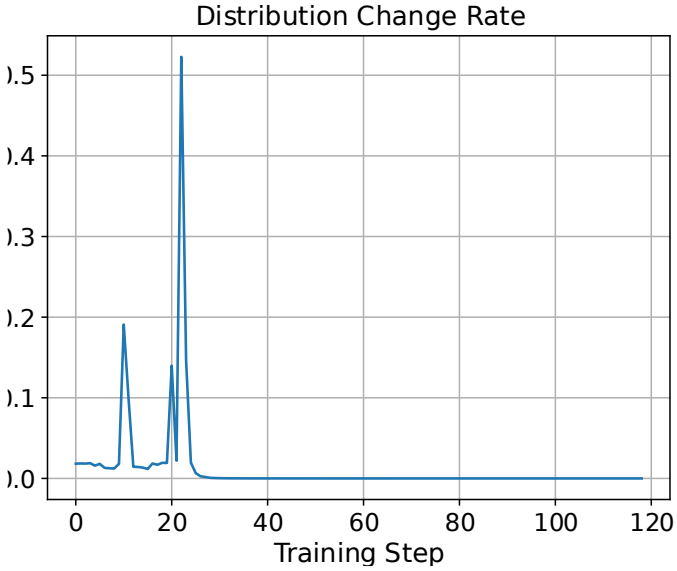




Time-Averaged Entropy vs Temperature
This plot shows the entropy, H, averaged over training steps as a function of softmax temperature. It characterizes how temperature controls distributional uncertainty, interpolating between low-entropy, sharply peaked predictions at low temperature and near-uniform distributions at high temperature.

Fixed-Temperature Loss vs Training Step
This plot shows the cross-entropy loss $-\ln p_{\text{target}}$ evaluated at a fixed temperature as a function of its training step. A decreasing loss indicates improved prediction of the target token, independent of changes in the full distribution entropy.





Sharpening Diagnostics: Maximum and Target Probability
This plot shows the evolution of the maximum predicted token probability and the target token probability as functions of training step. Increasing values indicate sharpening of the output distribution and improved target prediction, even when entropy decreases slowly.

Distribution Change Rate (KL Divergence)
This plot shows the Kullback–Leibler divergence between the model's output distributions at consecutive training steps. It quantifies the rate at which the predicted distribution changes, serving as a diagnostic for learning activity and convergence behavior.

Visual Demonstation of AI Learning Process – View 3

Process Documentation for Simulation

These are the steps/processes for the Simulation (Refer to the Mathcad Program that Follows):

1. Update memory $m \leftarrow \alpha m + x_t$.
2. Compute logits $\ell = W_O m$.
3. Convert to probabilities with Softmax $p(\ell/T)$.
4. Record entropy[1] $H = -\sum p \ln p$ and loss $L = -\ln p_{\text{target}}$.
5. Update weights using $W_O \leftarrow W_O - \epsilon(p - y)m^T$.
6. Sample next token $c_{t+1} \sim \text{Categorical}(p)$.

The core idea for this model comes from the Boltzmann Probability Law:

$$p_i = \frac{e^{\ell_i/T}}{\sum_j e^{\ell_j/T}}$$

Model Overview

We construct a minimal, discrete-state model that captures the essential probabilistic and learning dynamics of a GPT-style language model while remaining fully transparent and analytically interpretable. The model operates on a vocabulary of $N = 5$ symbolic tokens and evolves over discrete training steps $t = 0, \ldots,$steps $- 1$. Learning dynamics are explored across an ensemble of temperatures $T_k \in T_{\text{vec}}$, forming an $NT \times$ steps statistical ensemble.

State Representation and Memory Update

Each token is represented by a one-hot basis vector E_i. The model maintains a memory vector $m(t) \in \mathbb{R}^N$, updated according to

$$m(t+1) = \alpha^* m(t) + x_t$$

where $X_{i,t}$is the one-hot encoding of the token at step t, and $0 < \alpha < 1$controls exponential decay of past context. This memory plays the role of a coarse-grained contextual state.

Logits[2] and Softmax[3] Probabilities

At each step, logits are computed via a linear output head

$$\ell_i(t) = \sum_{j=0}^{N-1} W_{O,ij}(t)\, m_j(t)$$

These logits are converted into probabilities using a temperature-scaled Softmax*,

$$p_i(t; T_k) = \frac{\exp\left(\ell_i(t)/T_k\right)}{\sum_{j=0}^{N-1} \exp\left(\ell_j(t)/T_k\right)}$$

Where "Temperatures" $T_k \in T_{\text{vec}} = \{T_0, \ldots, T_{NT-1}\}$ controls stochasticity.
Low temperatures sharpen the distribution, while high temperatures approach a uniform distribution.
The softmax applies the standard exponential function to each element of the input tuple (consisting of real numbers), and normalizes these values by dividing by the sum of all these exponentials. The normalization ensures that the sum of the components of the output vector is 1.
The term "softmax" derives from the amplifying effects of the exponential on any maxima in the input tuple.

Entropy and Loss

The Shannon Entropy of the model's predictive distribution is computed as

$$H(T_k, t) = -\sum_{i=0}^{N-1} p_i(t; T_k) \ln p_i(t; T_k)$$

which quantifies uncertainty in the next-token prediction.
The training objective is the negative log-likelihood (cross-entropy loss) for a fixed target token $i =$ target,

$$L(T_k, t) = -\ln p_{\text{target}}(t; T_k)$$

Loss measures the surprise associated with the correct token and serves as the driving signal for learning.

Weight Update Rule
The output weights are updated using a gradient descent step derived from the softmax-cross-entropy objective,

$$W_{O,ij}(t+1) = W_{O,ij}(t) - \epsilon[p_i(t;T_k) - y_i]m_j(t)$$

where $y_i$is a one-hot target vector and $\epsilon$is the learning rate. This update increases the logit of the target token relative to others, reshaping the effective energy landscape.

Sampling Procedure
The next token is selected via true categorical (Boltzmann) sampling:

$$c_{t+1} \sim \text{Categorical}\,(p_i(t;T_k))$$

A deterministic pseudo-random table is used to generate reproducible draws. This corresponds to finite-temperature Boltzmann sampling rather than deterministic (argmax) decoding.

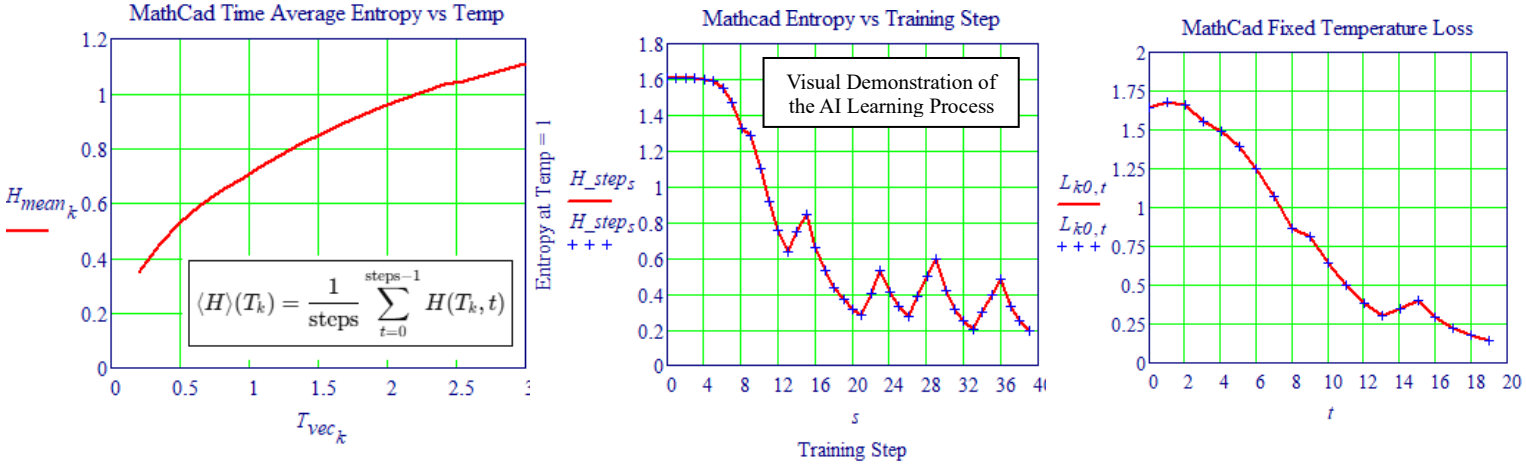
Ensemble Averaging
For each temperature T_k, entropy and loss are recorded across all steps. The time-averaged entropy is defined as

$$\langle H\rangle(T_k) = \frac{1}{\text{steps}} \sum_{t=0}^{\text{steps}-1} H(T_k, t)$$

This quantity provides a thermodynamic-style response function analogous to entropy–temperature relations in finite systems.

Visualization of the AI Learning Process

Entropy vs Training Step, $H(t)$
Shows the instantaneous Shannon entropy of the token distribution as learning progresses. Decreasing entropy indicates increasing confidence in predictions, while fluctuations reflect stochastic sampling and evolving context. These curves demonstrate that the model is learning. This AI process has decreased entropy or disorder.



Loss vs Training Step, $L(t)$ (See Below)
Displays the negative log-likelihood of the target token over training steps. Declining loss signifies successful learning, while transient increases indicate exploration or context shifts due to sampling.

Time-Averaged Entropy vs Temperature, $\langle H\rangle(T)$ (See Below)
Presents the ensemble-averaged entropy as a function of temperature. Low-temperature regimes correspond to ordered, low-entropy behavior, while high temperatures approach the maximum entropy limit $\ln N$, demonstrating phase-like crossover behavior in this finite system.

Dictionary:

Entropy is a measure of uncertainty or disorder, while information is the reduction of that uncertainty; high entropy means low information (more possibilities), and gaining information decreases entropy (reduces possibilities). In physics, information loss happens when a system's state becomes less defined (higher entropy), but the total information in the universe is conserved, even if it seems lost to us. Information can be seen as negative entropy because it adds structure and reduces randomness, but processes like data compression or measurement can transform information into less accessible (higher entropy) forms, leading to perceived information loss.

Key Concepts

Entropy (Information Theory**):** Measures the average uncertainty or unpredictability of a message or random variable.

High Entropy: High uncertainty, many possibilities, little information known (e.g., a truly random string of numbers).

Low Entropy: Low uncertainty, few possibilities, more information known (e.g., a repeating sequence).

Information: The capacity to reduce uncertainty. Gaining information lowers entropy.

Information Loss: Occurs when a process makes a system less distinguishable, increasing its entropy (e.g., a system evolving from a specific state to a mixed state).

Model: This model contains explicit query, key, value, and output projections, reproducing the essential mechanism by which transformer attention weights contextual information.

References - Hyperlinks

1. *A Characterization of Entropy in Terms of Information Loss*, John C. Baez,
2. "Logit Function" Wikipedia
3. "Softmax Function" Wikipedia
4. "Shannon Entropy" Wikipedia
5. "Attention Layer" "The Hundred-Page Language Models Book", Andriy Burkov
   Chapter 4, Attention Layer: The self-attention layer, transforms each input embedding vector **x**E into a new vector **g**E for every token t, from 1 to L, where L represents the input length.

Dynamics and Summary of Math Model

The model operates on a finite vocabulary of size $N = 60$. Each token is represented by a one-hot vector $\mathbf{e}_i \in \mathbb{R}^N$, and the model state at training step t is a context matrix $\mathbf{X}(t) \in \mathbb{R}^{N\times L}$with $L =$ 80 columns, each column corresponding to a token in the rolling context window. The query token is the most recent element of the context, $\mathbf{x}_q = \mathbf{x}_L$. This representation corresponds to a discrete set of **Statistical Mechanics Microstates** in a finite configuration space.

Learned linear projections define query, key, and value representations through matrices $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V \in \mathbb{R}^{N\times N}$. The query vector is $\mathbf{q} = \mathbf{W}_Q\mathbf{x}_q$, while keys and values are $\mathbf{K} = \mathbf{W}_K\mathbf{X}$and $\mathbf{V} = \mathbf{W}_V\mathbf{X}$. These projections define an interaction basis in which token–token correlations are evaluated, analogous to transforming into an interaction frame in physics.

Scaled Dot-Product Attention is computed via scores $s_j = \mathbf{q}^\top\mathbf{k}_j/\sqrt{N}$for each context position j. The attention weights are $a_j = \exp(s_j)/\sum_m \exp(s_m)$, and the attention context vector is $\mathbf{c}_{\text{att}} = \sum_j a_j\,\mathbf{v}_j$. This mechanism is equivalent to a Boltzmann-weighted mean-field interaction, with the $\sqrt{N}$factor playing the role of an energy normalization.

A residual connection combines the query and attention context as $\mathbf{u}_1 = \mathbf{x}_q + \mathbf{c}_{\text{att}}$.
Layer normalization is applied as $\text{LN}(\mathbf{u}) = (\mathbf{u} - \langle\mathbf{u}\rangle)/\sqrt{\langle(\mathbf{u} - \langle\mathbf{u}\rangle)^2\rangle}$, producing $\mathbf{h}_1 = \text{LN}(\mathbf{u}_1)$. This operation removes the mean and fixes the variance of the state, analogous to imposing a microcanonical constraint on the system.

Nonlinear feature recombination is introduced through a feed-forward network with weights $\mathbf{W}_1 \in \mathbb{R}^{2N\times N}$and $\mathbf{W}_2 \in \mathbb{R}^{N\times 2N}$. The transformation is $\mathbf{z} = \tanh(\mathbf{W}_1\mathbf{h}_1)$, followed by $\mathbf{f} = \mathbf{W}_2\mathbf{z}$. A second residual connection and layer normalization yield the final hidden state $\mathbf{h}_2 = \text{LN}(\mathbf{h}_1 + \mathbf{f})$. This stage acts as a local nonlinear scattering process that redistributes information across features.

The output logits are computed using an output projection $\mathbf{W}_O \in \mathbb{R}^{N\times N}$as $\boldsymbol{\ell} = \mathbf{W}_O\mathbf{h}_2$. The training probability distribution is obtained via **softmax** at unit temperature, $p_i = \exp(\ell_i)/\sum_j \exp(\ell_j)$, while evaluation at arbitrary temperature T uses $p_i(T) = \exp(\ell_i/T)/\sum_j \exp(\ell_j/T)$. This construction directly parallels a canonical ensemble with temperature controlling distributional sharpness.

Given a target token index i^*, **the loss is defined** as $L = -\ln p_{i^*}$.
The entropy of the output distribution is $H(T) = -\sum_i p_i(T)\ln p_i(T)$, with a maximum value $H_{\max} = \ln N$ corresponding to a uniform distribution. The *loss measures predictive accuracy* for the target state, while the *entropy measures global uncertainty across all states*.

Learning proceeds via gradient descent. The **gradient signal** at the output is $g_i = p_i - \delta_{ii^*}$, and the output weights are updated as $\mathbf{W}_O \leftarrow \mathbf{W}_O - \eta_O\mathbf{g}\mathbf{h}_2^\top$. An approximate back-propagated signal $\boldsymbol{\delta} = \mathbf{W}_O^\top\mathbf{g}$ drives slower updates to the attention projections, with $\mathbf{W}_Q \leftarrow \mathbf{W}_Q - \eta_{QKV}\boldsymbol{\delta}\mathbf{x}_q^\top$and $\mathbf{W}_K, \mathbf{W}_V$updated by **attention-weighted sums** over context tokens. This separation of learning rates reflects a hierarchy in which output mappings adapt faster than interaction geometry.
The context evolves deterministically by selecting the most probable token $i_{\text{next}} = \arg\max_i p_i$, shifting the context window, and appending the corresponding one-hot vector.

This produces a trajectory toward low-entropy attractors rather than stochastic exploration.

Diagnostic Quantities (See the Above Plots):
include the training entropy $H_{\text{train}}(t) = -\sum_i p_i \ln p_i$, the Kullback–Leibler divergence between successive distributions $D_{\text{KL}}(t) = \sum_i p_i(t)\ln[p_i(t)/p_i(t-1)]$, and the maximum predicted probability $p_{\max}(t) = \max_i p_i(t)$. Together, these quantities describe a finite, non-equilibrium information system in which attention defines effective interactions, normalization enforces conserved scale, cross-entropy loss injects directed free energy, and training drives the system from a high-entropy state toward structured, low-entropy configurations.

Python Code for Transformer Model

```
import os
import numpy as np
import matplotlib
import matplotlib.pyplot as plt

# GLOBAL PLOT SETTINGS
FONT_SCALE = 1.35
BASE_FONT = 10

matplotlib.rcParams.update({
    "font.size": BASE_FONT * FONT_SCALE,
    "axes.titlesize": BASE_FONT * 1.1 * FONT_SCALE,
    "axes.labelsize": BASE_FONT * 1.1 * FONT_SCALE,
    "xtick.labelsize": BASE_FONT * FONT_SCALE,
    "ytick.labelsize": BASE_FONT * FONT_SCALE,
    "legend.fontsize": BASE_FONT * FONT_SCALE,
    "figure.titlesize": BASE_FONT * 1.2 * FONT_SCALE,
    "svg.fonttype": "none",
})

# Utilities
def softmax(z):
    z = z - np.max(z)
    ez = np.exp(z)
    return ez / np.sum(ez)

def layer_norm(x, eps=1e-8):
    mu = np.mean(x)
    var = np.mean((x - mu) ** 2)
    return (x - mu) / np.sqrt(var + eps)

def kl_divergence(p, q, eps=1e-12):
    return np.sum(p * np.log((p + eps) / (q + eps)))

def get_output_dir():
    try:
        base = os.path.dirname(os.path.abspath(__file__))
    except NameError:
        base = os.getcwd()
    outdir = os.path.join(base, "N80_FINAL_OUTPUTS_2X
    os.makedirs(outdir, exist_ok=True)
    return outdir

def save_svg(fig, outdir, name):
    path = os.path.join(outdir, name)
    fig.savefig(path, format="svg")
    plt.close(fig)
    print("Saved:", path)

# Main Program
def main():

    outdir = get_output_dir()
    print("Saving outputs to:", outdir)

    # -------------------------
    # Parameters
    # -------------------------
    N = 80
    steps = 60        # <<< CHANGED from 120
    NT = 30

    Lctx = 8
    d = N
    d_ff = 2 * d

    lr_out = 3e-2
    lr_qkv = 1e-3
    Tmin, Tmax = 0.2, 3.5
    Tvec = np.linspace(Tmin, Tmax, NT)

    Ttrain = 1.0
    Tsample = 0.0

    target = 10
    k0 = int(np.argmin(np.abs(Tvec - 1.0)))

    rng = np.random.default_rng(12345)
    E = np.eye(N)

    # Storage
    H = np.zeros((NT, steps))
    L = np.zeros((NT, steps))
    KL = np.zeros(steps - 1)

    Htrain = np.zeros(steps)
    pmax = np.zeros(steps)
    ptarget = np.zeros(steps)

    ATT = np.zeros((steps, Lctx))

    # -------------------------
    # Initialize weights
    # -------------------------
    WO = 0.05 * (rng.random((N, d)) - 0.5)
    WQ = 0.05 * (rng.random((d, N)) - 0.5)
    WK = 0.05 * (rng.random((d, N)) - 0.5)
    WV = 0.05 * (rng.random((d, N)) - 0.5)
    W1 = 0.05 * (rng.random((d_ff, d)) - 0.5)
    W2 = 0.05 * (rng.random((d, d_ff)) - 0.5)

    # -------------------------
    # Context
    # -------------------------
    Xctx = np.zeros((N, Lctx))
    Xctx[:, -1] = E[:, 0]

    prev_p_train = None

    # Training loop

    for t in range(steps):

        Kmat = WK @ Xctx
        Vmat = WV @ Xctx
        xq = Xctx[:, -1].reshape(d, 1)
        q = WQ @ xq

        scores = (q.T @ Kmat).flatten() / np.sqrt(d)
        a = softmax(scores)
        ATT[t] = a

        c_att = (Vmat * a).sum(axis=1).reshape(d, 1)
        c_res1 = layer_norm(xq + c_att)

        h = np.tanh(W1 @ c_res1)
        ffn_out = W2 @ h
        c_final = layer_norm(c_res1 + ffn_out)

        logits = (WO @ c_final).flatten()
        p_train = softmax(logits / Ttrain)

        Htrain[t] = -np.sum(p_train * np.log(p_train + 1e-300))
```

```
    pmax[t] = np.max(p_train)
    ptarget[t] = p_train[target]

    if prev_p_train is not None:
        KL[t - 1] = kl_divergence(p_train, prev_p_train)
    prev_p_train = p_train.copy()

    for k in range(NT):
        p_k = softmax(logits / Tvec[k])
        H[k, t] = -np.sum(p_k * np.log(p_k + 1e-300))
        L[k, t] = -np.log(p_k[target] + 1e-300)

    g = p_train.copy()
    g[target] -= 1.0
    WO -= lr_out * (g.reshape(N, 1) @ c_final.T)

    delta = (WO.T @ g.reshape(N, 1))
    WQ -= lr_qkv * (delta @ xq.T)
    for j in range(Lctx):
        xj = Xctx[:, j].reshape(N, 1)
        WK -= lr_qkv * (delta @ xj.T) * a[j]
        WV -= lr_qkv * (delta @ xj.T) * a[j]

    if t < steps - 1:
        kk = int(np.argmax(p_train))
        Xctx[:, :-1] = Xctx[:, 1:]
        Xctx[:, -1] = E[:, kk]
# Save TXT
np.savetxt(os.path.join(outdir, "L_python.txt"), L)
np.savetxt(os.path.join(outdir, "Tvec_python.txt"), Tvec.reshape(-1, 1))
np.savetxt(os.path.join(outdir, "Htrain_python.txt"), Htrain)
# SVG Plots (THICK LINES)
s = np.arange(steps)

fig = plt.figure()
plt.plot(s, Htrain, linewidth=3.0)
plt.xlabel("Training Step")
plt.ylabel("Entropy H (Ttrain=1)")
plt.title("Training Entropy vs Step (N=80)")
plt.grid(True)
save_svg(fig, outdir, "Htrain_vs_step.svg")

fig = plt.figure()
plt.plot(s, H[k0], linewidth=3.0)
plt.xlabel("Training Step")
plt.ylabel("Entropy H")
plt.title(f"Entropy vs Step at T={Tvec[k0]:.2f}")
plt.grid(True)
save_svg(fig, outdir, "H_vs_training_step.svg")

fig = plt.figure()
plt.plot(s, L[k0], linewidth=3.0)
plt.xlabel("Training Step")
plt.ylabel("Loss")
plt.title(f"Loss vs Step at T={Tvec[k0]:.2f}")
plt.grid(True)
save_svg(fig, outdir, "FixedTempLoss.svg")

fig = plt.figure()
plt.plot(Tvec, H.mean(axis=1), linewidth=3.0)
plt.xlabel("Temperature T")
plt.ylabel("⟨H⟩")
plt.title("Time-Averaged Entropy vs Temperature")
plt.grid(True)
save_svg(fig, outdir, "TimeAvg_H_vs_Temp.svg")

fig = plt.figure()
plt.plot(H[k0], L[k0], marker="o", ms=4, linewidth=2.0)
    plt.xlabel("Entropy H")
    plt.ylabel("Loss L")
    plt.title("Entropy–Loss Phase Trajectory")
    plt.grid(True)
    save_svg(fig, outdir, "Entropy_vs_Loss.svg")

    fig = plt.figure()
    plt.plot(s[:-1], KL, linewidth=3.0)
    plt.xlabel("Training Step")
    plt.ylabel("KL[p(t)||p(t−1)]")
    plt.title("Distribution Change Rate")
    plt.grid(True)
    save_svg(fig, outdir, "KL_vs_training_step.svg")

    fig = plt.figure()
    plt.plot(np.arange(Lctx), ATT.mean(axis=0),
marker="o", ms=5, linewidth=2.0)
    plt.xlabel("Context Position")
    plt.ylabel("⟨Attention Weight⟩")
    plt.title("Mean Attention Profile")
    plt.grid(True)
    save_svg(fig, outdir, "Attention_vs_Position.svg")

    fig = plt.figure()
    plt.plot(s, ptarget, label="p(target)", linewidth=3.0)
    plt.plot(s, pmax, label="p(max)", linewidth=3.0)
    plt.xlabel("Training Step")
    plt.ylabel("Probability")
    plt.title("Sharpening Diagnostics")
    plt.legend()
    plt.grid(True)
    save_svg(fig, outdir, "pmax_ptarget_vs_step.svg")

    print("\nDONE — 60 steps, doubled plot line
thickness.")

if __name__ == "__main__":
    main()
```