Who Needs Attention Anyway?
Elastic State Models as Geometry-Aware Adaptive Compute

Dario Fumarola

Amazon Web Services - Prototyping and Customer Engineering

The Attention Tax

Attention's power comes with a **tax**: at every step it scans a window of past states, so **compute scales with context length** rather than **how difficult the current state is**.

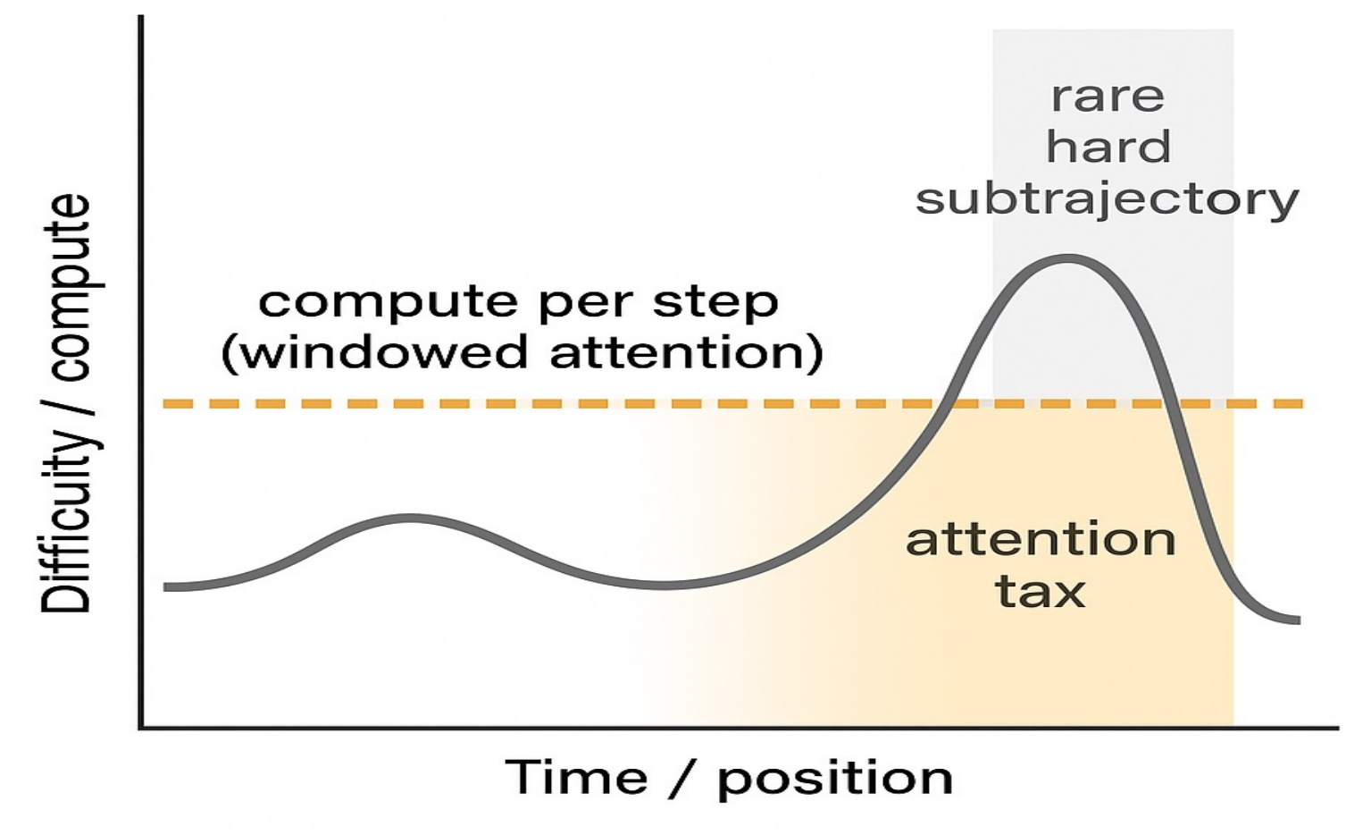

Figure 1. Attention tax vs task difficulty.

Long stretches of smooth dynamics are charged just as heavily as the rare junctions, shocks, and clashes where small errors change the outcome. What we really want is a model that **saves effort on easy steps** and **spends it on those high-stakes moments**.

From Flat to Elastic Compute

Instead of using the same compute at every step, we **let the model vary how much it thinks** from state to state.

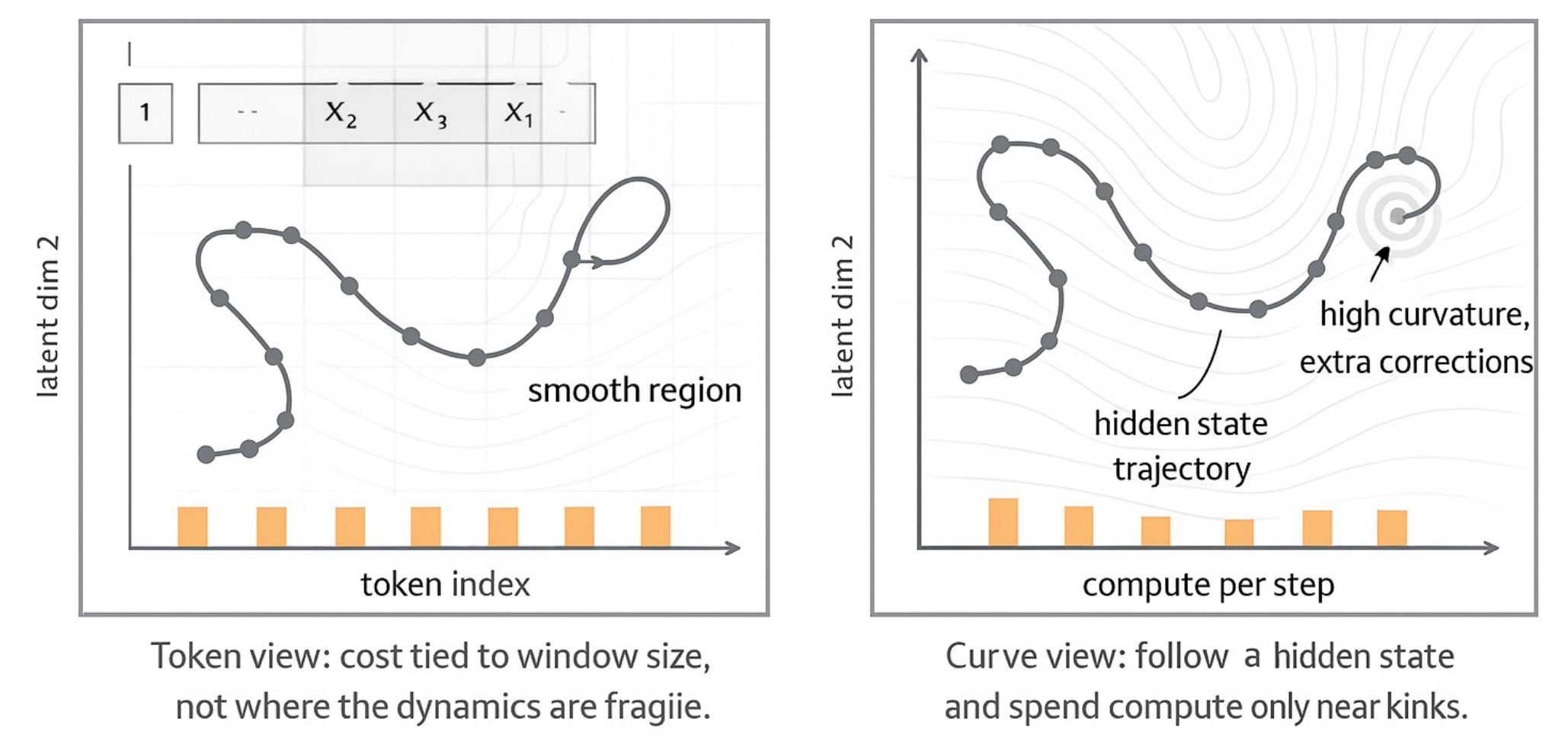
1
X2
X3
X1
latent dim 2
smooth region
token index
Token view: cost tied to window size, not where the dynamics are fragiie.
latent dim 2
high curvature, extra corrections
hidden state trajectory
compute per step
Curve view: follow a hidden state and spend compute only near kinks.

Figure 2. Elastic Compute along a Hidden-State Trajectory.

- **Per-step difficulty.** Each new state gets a simple *"how wrong does this look?"* score from the same signal that trains the model (e.g., value error or energy).
- **Per-step budget.** A small gate turns that score into a small integer budget, deciding whether this step is good enough or **deserves a few extra internal updates**.

Thinking in Curves, Not Tokens

In streaming settings, the core object isn't a list of past tokens; **it's a hidden state tracing a curve over time**. The question stops being *"how big should my window be?"* and becomes "*where along this trajectory is the system actually fragile?"*

- Most of the curve is smooth, with occasional sharp bends at junctions, contacts, or regime shifts. A good model should **save effort on straight segments** and **spend it around those kinks where small deviations** really matter.
- In RL, control, and physical dynamics, this curve lives in *latent space*, but its geometry is real: flat stretches, bottlenecks, and cliffs. Thinking in curves means **using compute to keep that path stable and safe**, instead of repeatedly sweeping a fixed attention window.

Elastic State Model (ESM)

An **Elastic State Model (ESM)** turns the flat-vs-elastic idea into a concrete layer on top of a streaming state-space model.

1. **Streaming backbone and per-step objective.** A cheap SSM maintains the latent state z_t and proposes the next state z^0_{t+1}. The same per-step loss used for training is attached and remains well-defined at test time.
2. **Geometry-aware latent corrector.** When we decide to spend extra compute, ESM takes **gradients of that loss with respect to the latent state**, passes them through a **learned SPD metric**, and applies a small number of **trust-region–clipped updates**, with a drift term to keep corrections local and stable.
3. **Gated depth under a budget.** The per-step difficulty and budget are implemented as a tiny gate that chooses an inner depth $K_t \in \{0, \dots, K_{max}\}$. It is trained with an explicit compute penalty $\lambda \mathbb{E}[K_t]$, so extra steps are only used when they materially improve the task.

Stepwise Elastic Compute Block

At each timestep, the streaming backbone proposes a **next state and a per-step error**. A small gate turns this error into a **difficulty score** and either accepts the proposal or routes the state through an extra correction block before committing the next state.

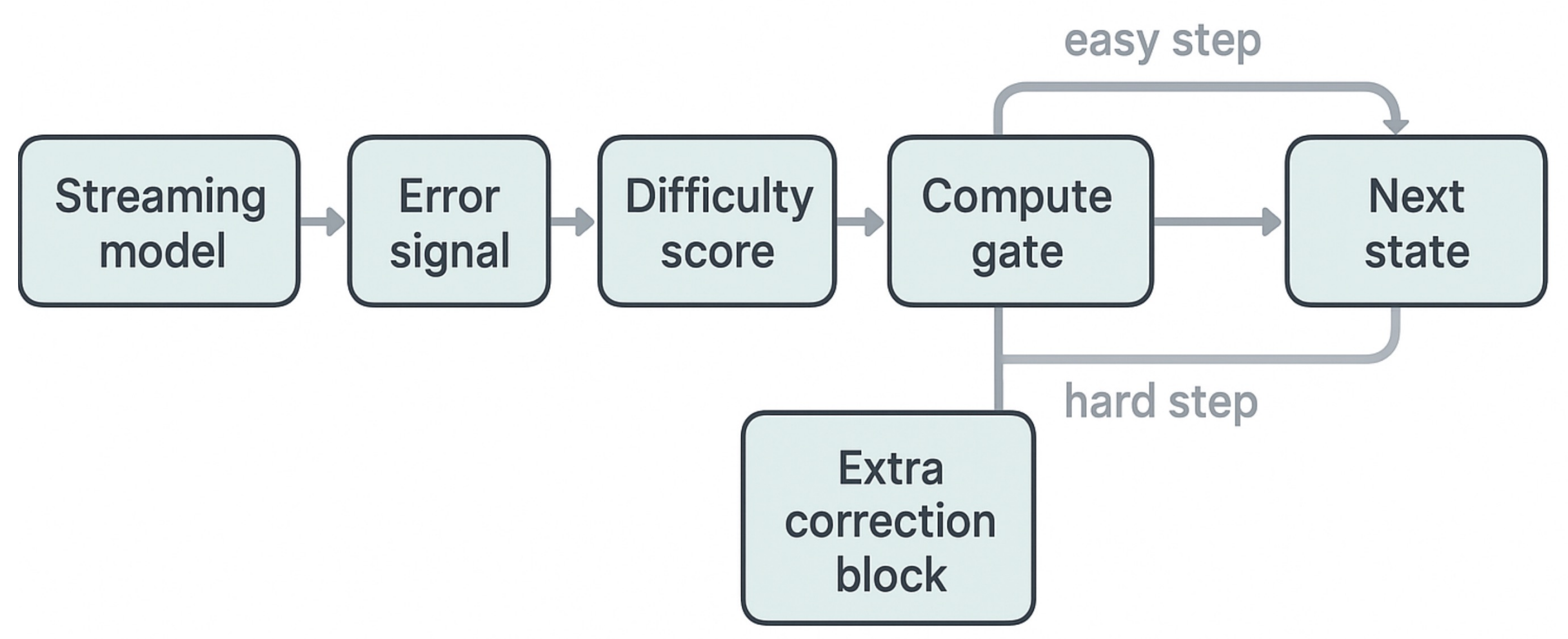

Figure 3. Stepwise Elastic Compute Block: a difficulty gate routes hard steps through a local correction module.

Latent Geometry

When a correction step is taken, we operate locally on the backbone's proposed state in latent space. At step t, the backbone produces

$$z^0_{t+1} = f_\theta(z_t, x_{t+1}), \quad y^0_{t+1} = g_\phi(z^0_{t+1}), \quad \ell_t = L(y^0_{t+1}).$$

We treat ℓ_t as a local objective to be reduced by a small updates around z^0_{t+1}.

Metric-weighted residual and step.

We take the gradient $g_t = \nabla_{z^0_{t+1}} \ell_t$

and pass it through a learned SPD metric G_ψ. This defines a scalar $\kappa_t = g_t^\top G_\psi^{-1} g_t$

and a preconditioned direction $v_t = -G_\psi^{-1} g_t$.

We clip $|v_t|$ in the metric norm using a trust-region radius that can depend on κ_t, then update $z_{t+1} = \mathrm{Retract}\big(z^0_{t+1} + \alpha v_t\big)$.

Maze and Protein Repair

We plug the same elastic compute block into two very different streaming systems and visualize where the extra inner steps goes.

- **Maze agent.** An agent moves step by step through a grid maze with a streaming backbone state; maze layouts change between episodes, so it can't just memorize the map. Along long straight corridors the gate keeps $K \approx 0$, but near junctions, doors, and tight gaps the error spikes and the gate turns on extra inner steps, concentrating compute at bottlenecks.
- **Protein loop repair.** We take a folded protein, bend one loop into a bad shape, and refine in latent space. Extra compute clusters on residues near the distorted loop while the rest of the backbone stays near SSM cost, driving energy down faster than either a uniform high-compute refiner or a Transformer at similar budget.

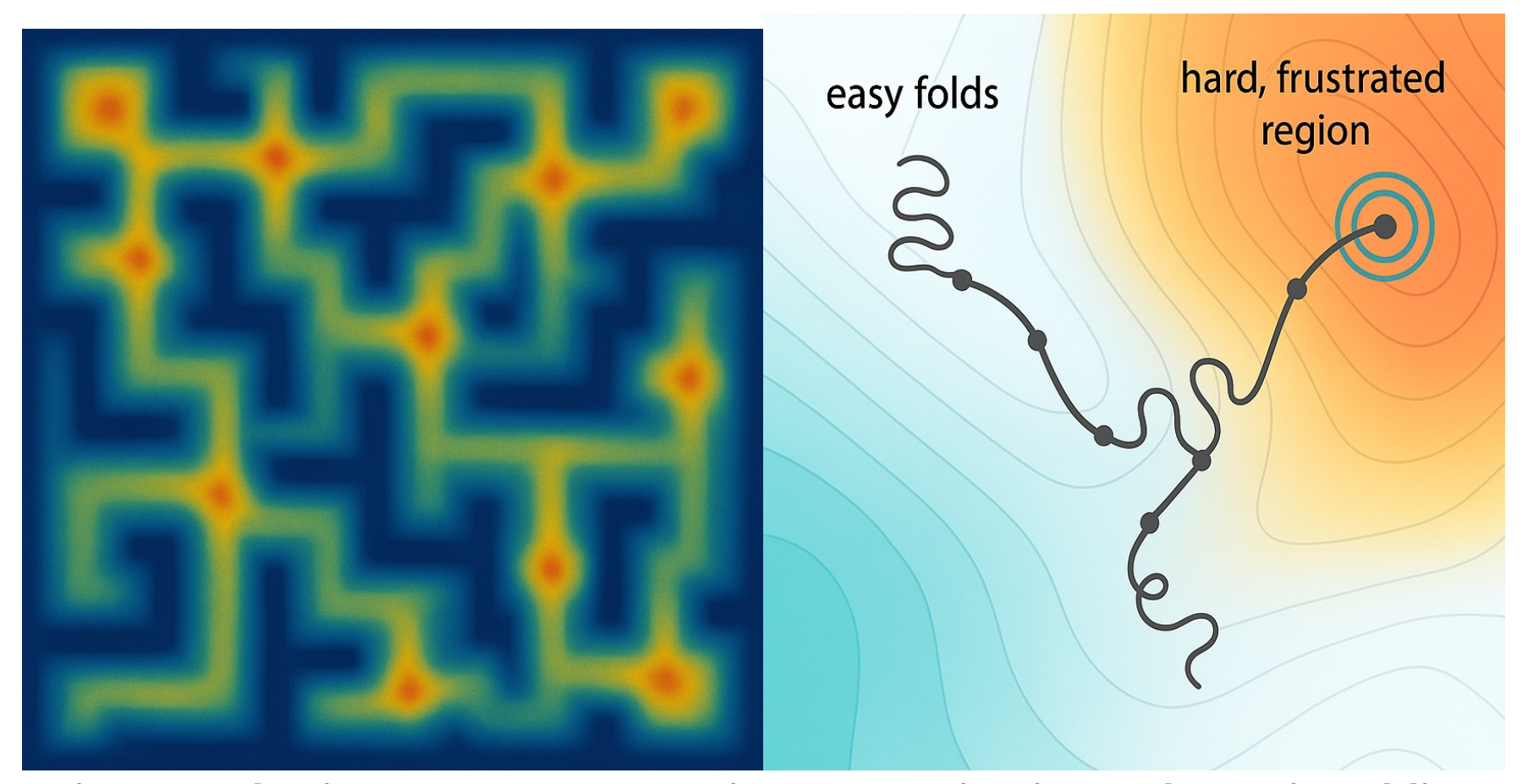

Figure 4. Elastic Compute Hotspots in Maze Navigation and Protein Folding.

Compute-Matched Results

We compare four agents: a **plain streaming SSM**, a **standard Transformer**, an **SSM with a windowed-attention** head, and our **Elastic State Model (ESM)**. Maze success and protein energy are normalized to the SSM baseline; average compute is normalized so the Transformer has cost 1.0.

Method	Maze success ↑	Protein ΔEnergy ↓	Avg compu
SSM	0.78	1.00	0.45
Transformer	0.93	0.76	1.00
SSM + attentior	0.91	0.80	0.95
ESM (ours)	0.96	0.68	0.78

Table 1. Elastic State Models vs Attention at Matched Compute.

ESM closes the gap between a plain SSM and a full Transformer without paying the flat attention tax on every step. Instead of spending the same heavy budget everywhere, the gate concentrates geometric updates on the rare, high-impact states identified by the loss, achieving **Transformer-level performance at similar compute**.

References

[1] Albert Gu, Karan Goel, and Christopher Ré.
Efficiently Modeling Long Sequences with Structured State Spaces.
International Conference on Learning Representations (ICLR), 2022.

[2] Alex Graves.
Adaptive Computation Time for Recurrent Neural Networks.
arXiv preprint arXiv:1603.08983, 2016.