Transition
into the Meta-Learning land

Scroll down for pre-2000 papers. See NIPS 2016 slides or similar NIPS 2018 WS slides.

Jürgen Schmidhuber's page on

LEARNING TO LEARN

METALEARNING MACHINES AND RECURSIVE SELF-IMPROVEMENT

Gödel machine paper: J. Schmidhuber. Ultimate Cognition *à la* Gödel. *Cognitive Computation* 1(2):177-193, 2009. PDF. (Springer.) More papers on Gödel machines.

Overview article: T. Schaul and J. Schmidhuber. Metalearning. Scholarpedia, 5(6):4650, 2010.

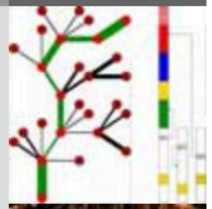
OOPS paper: J. Schmidhuber. Optimal Ordered Problem Solver. *Machine Learning,* 54, 211-254, 2004. PDF. HTML. HTML overview.

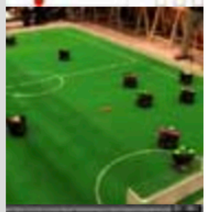
More on meta-learning Gödel machines and

Most machine learning researchers focus on domain-specific learning algorithms. Can we also construct general purpose learning algorithms, in particular, meta-learning algorithms that can learn better learning algorithms? This question has been a main drive of Schmidhuber's research since his diploma thesis on metalearning in 1987 [1], where he applied Genetic Programming (GP) to itself, to recursively evolve better GP methods.

Metalearning (or Meta-Learning) means learning the credit assignment method itself through self-modifying code. Metalearning may be the most ambitious but also the most rewarding goal of machine learning. There are few limits to what a good metalearner will learn. Where appropriate it will learn to learn by analogy, by chunking, by planning, by subgoal generation, by combinations thereof - you name it.

Some of Schmidhuber's earlier approaches to metalearning and general purpose learning machines are based on self-modifying policies (SMPs) and incremental self-improvement based on SMPs. The learning algorithm of an SMP is part of the SMP itself - SMPs can modify the way they modify themselves. Schmidhuber introduced several ways of forcing SMPs to come up with better and better self-modification algorithms: (a) The "success-story algorithm" [6-14] **(click here for talk slides),** (b) Gradient calculation in recurrent nets [2-5], (c) Market models of the mind





Most machine learning researchers focus on domain-specific learning algorithms. Can we also construct general purpose learning algorithms, in particular, meta-learning algorithms that can learn better learning algorithms? This question has been a main drive of Schmidhuber's research since his diploma thesis on metalearning in 1987 [1], where he applied Genetic Programming (GP) to itself, to recursively evolve better GP methods.

Metalearning (or Meta-Learning) means learning the credit assignment method itself through self-modifying code. Metalearning may be the most ambitious but also the most rewarding goal of machine learning. There are few limits to what a good metalearner will learn. Where appropriate it will learn to learn by analogy, by chunking, by planning, by subgoal generation, by combinations thereof - you name it.

Learner and Meta-Learner

Another popular view of meta-learning decomposes the model update into two stages:

- A classifier f_θ is the "learner" model, trained for operating a given task;
- In the meantime, a optimizer g_ϕ learns how to update the learner model's parameters via the support set S, $\theta' = g_\phi(\theta, S)$.

Then in final optimization step, we need to update both θ and ϕ to maximize:

$$\mathbb{E}_{L \subset \mathcal{L}}[\mathbb{E}_{S^L \subset \mathcal{D}, B^L \subset \mathcal{D}}[\sum_{(\mathbf{x}, y) \in B^L} P_{g_\phi(\theta, S^L)}(y|\mathbf{x})]]$$

Fast weight programming

Before diving into the delta rule, let's first review fast weight programming, which will help us understand the delta rule and other test-time-trainers (TTT, Titans, Mesa layer, etc.).

> "Fast weights provide a neurally plausible way of implementing the type of temporary storage that is required by working memory, while slow weights capture more permanent associations learned over many experiences." – Geoffrey Hinton

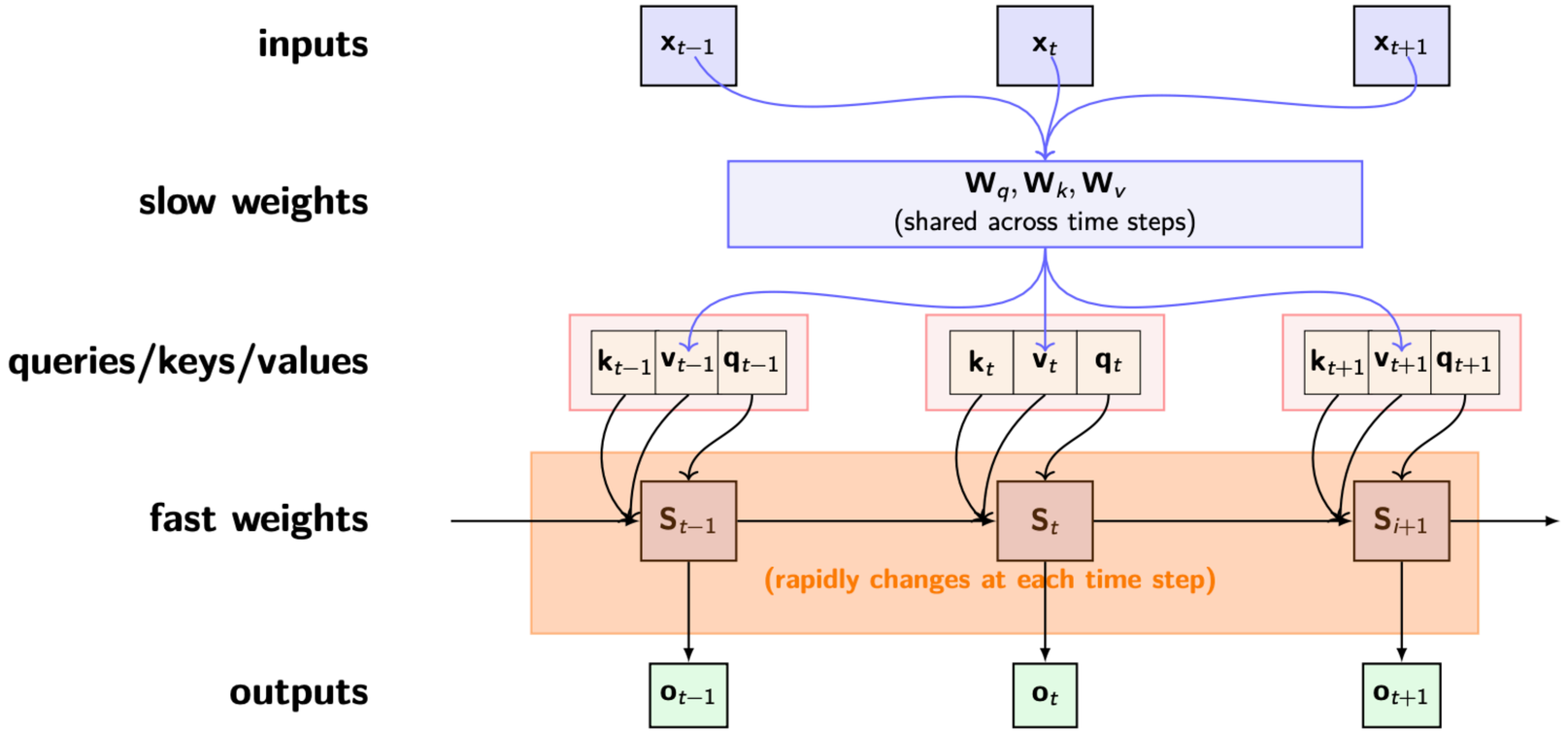
Recall the memory readout in linear attention:

$$\mathbf{o}_t = \mathbf{S}_t \mathbf{q}_t$$

We can think of the recurrent hidden state $\mathbf{S}_t$ as a fast weight matrix that maps input $\mathbf{q}_t$ to output $\mathbf{o}_t$ and is updated as it goes:

$$\mathbf{S}_t = \mathbf{S}_{t-1} + \mathbf{v}_t \mathbf{k}_t^\top$$

Linear attention is secretly a fast weight programmer



- **Fast Weight**: $\mathbf{S}_t$ maps $\mathbf{q}_t$ to $\mathbf{o}_t$, updated dynamically during inference for rapid adaptation.
- **Slow Weight**: $\mathbf{W}_q$, $\mathbf{W}_k$, and $\mathbf{W}_v$ are fixed during inference and only updated during training (e.g., via gradient descent).

The choice of update rule

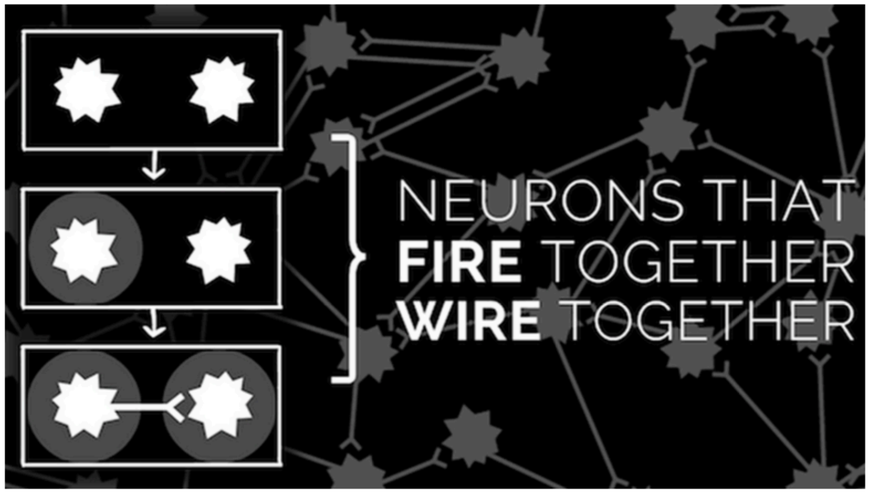


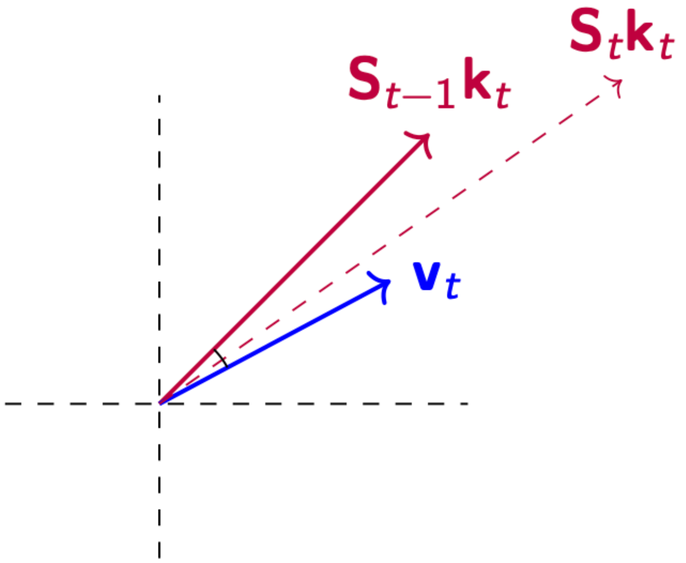
Figure: The principle of Hebbian learning.

- Hebbian update rule: $\mathbf{S}_t = \mathbf{S}_{t-1} + \mathbf{v}_t \mathbf{k}_t^\top$
- Delta rule: $\mathbf{S}_t = \mathbf{S}_{t-1} - \beta_t \left(\mathbf{S}_{t-1}\mathbf{k}_t - \mathbf{v}_t\right) \mathbf{k}_t^\top$
- ...

Both Hebbian and delta update rules can be regarded as optimizing online learning objective via test-time SGD.

Linear Attention: Test-Time Objective

Maximize alignment
$=$ Minimize angle difference $+$ enlarge $|\mathbf{Sk}|$

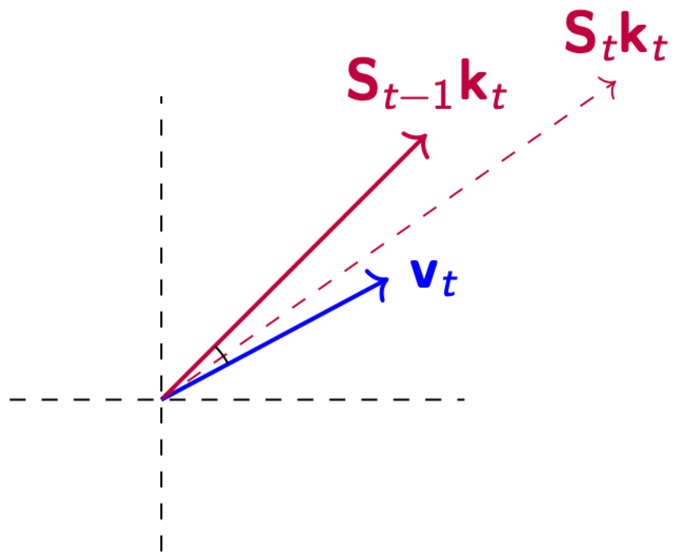


Objective: $\mathcal{L}_t(\mathbf{S}) = -\langle \mathbf{S}\mathbf{k}_t, \mathbf{v}_t \rangle$

SGD update: $\mathbf{S}_t = \mathbf{S}_{t-1} - \beta_t \nabla \mathcal{L}_t(\mathbf{S}_{t-1}) = \mathbf{S}_{t-1} + \beta_t \mathbf{v}_t \mathbf{k}_t^\top$

Linear Attention: Test-Time Objective

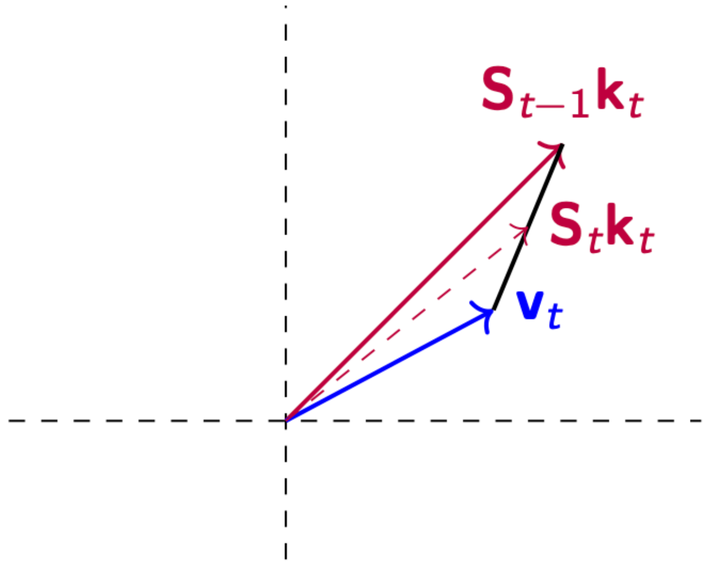
Maximize alignment
$=$ Minimize angle difference $+$ enlarge $|\mathbf{Sk}|$



- Linear attention may favor increasing $|\mathbf{Sk}|$ over angle alignment, leading to numerical instabilities.
- Mamba2 uses decay ($\mathbf{S}_t = \alpha_t \mathbf{S}_{t-1} + \mathbf{v}_t \mathbf{k}_t^\top$) to control $|\mathbf{Sk}|$, stabilizing the training process.

DeltaNet: Test-Time Objective

Directly minimize Euclidean distance



Objective: $\mathcal{L}_t(\mathbf{S}) = \frac{1}{2}\|\mathbf{S}\mathbf{k}_t - \mathbf{v}_t\|^2$

SGD update: $\mathbf{S}_t = \mathbf{S}_{t-1} - \beta_t \nabla \mathcal{L}_t(\mathbf{S}_{t-1}) = \mathbf{S}_{t-1} - \beta_t(\mathbf{S}_{t-1}\mathbf{k}_t - \mathbf{v}_t)\mathbf{k}_t^\top$

- Better numerical stability: the norm of $\mathbf{S}_t$ is controlled.
- Better in-context associative recall: directly optimizes key-value association prediction (Liu et al. 2024)

In-context associative recall

Multi-Query Associative Recall (MQAR, Arora et al. 2023)

A synthetic benchmark for testing in-context associative recall. **Example:**

- Given key-value pairs: "A 4 B 3 C 6 F 1 E 2"
- Query: "A ? C ? F ? E ? B ?"
- Expected output: "4, 6, 1, 2, 3"

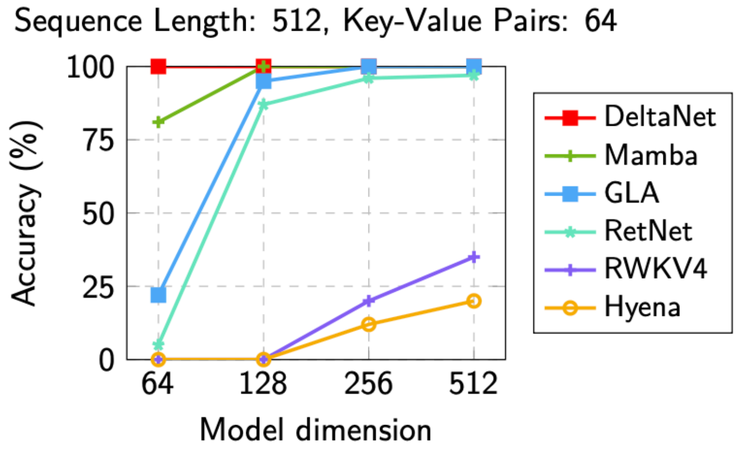


Figure: Accuracy (%) on MQAR. DeltaNet achieves the perfect recall.

Beyond linear regression objective

DeltaNet optimizes the online linear regression loss:

$$\mathcal{L}_t(\mathbf{S}) = \frac{1}{2}\|\mathbf{S}\mathbf{k}_t - \mathbf{v}_t\|^2$$

- This optimization objective assumes linear relationships in historical data dependencies
- However, generative AI tasks involve complex, nonlinear dependencies
- A linear regression loss may be insufficient to capture these rich patterns.

Going beyond online linear regression objective

TTT (Sun et al. 2024a) extends this to a nonlinear regression loss:

$$\mathcal{L}_t(\mathbf{S}) = \frac{1}{2}\|f_{\mathbf{S}}(\mathbf{k}_t) - \mathbf{v}_t\|^2$$

where $f_{\mathbf{S}}$ is a nonlinear transformation parameterized by $\mathbf{S}$.
Examples:

- TTT-linear:

$$f_{\mathbf{S}}(x) = \mathrm{LN}(\mathbf{S}x) + x,$$

- TTT-MLP:

$$f_{\mathbf{S}}(x) = \mathrm{LN}(\mathrm{MLP}_{\mathbf{S}}(x)) + x$$

where LN denotes layer normalization.

This is actually useful!

One-Minute Video Generation with Test-Time Training

Karan Dalal*[4] Daniel Koceja*[2] Gashon Hussein*[2] Jiarui Xu*[1,3] Yue Zhao†[5] Youjin Song†[2]
Shihao Han[1] Ka Chun Cheung[1] Jan Kautz[1] Carlos Guestrin[2] Tatsunori Hashimoto[2] Sanmi Koyejo[2]
Yejin Choi[1] Yu Sun[1,2] Xiaolong Wang[1,3]
[1]NVIDIA [2]Stanford University [3]UCSD [4]UC Berkeley [5]UT Austin

Figure 1. TTT layers enable a pre-trained Diffusion Transformer to generate one-minute videos from text storyboards. We use *Tom and Jerry* cartoons as a proof of concept. The videos tell complex stories with coherent scenes composed of dynamic motion. Every video is produced directly by the model in a single shot, without editing, stitching, or post-processing. Every story is newly created.

https://test-time-training.github.io/video-dit/

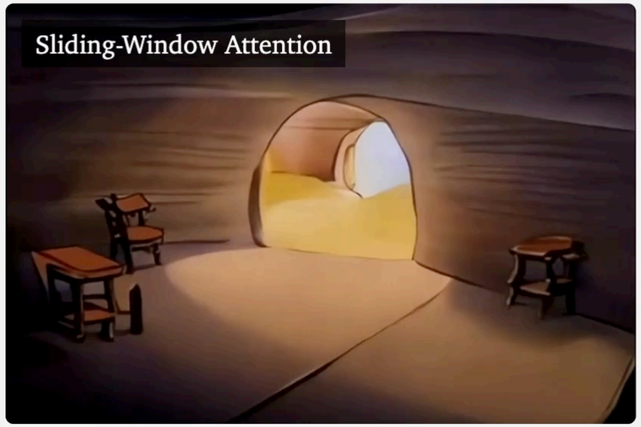
Baseline Comparisons

TTT-MLP outperforms all other baselines in temporal consistency, motion smoothness, and overall aesthetics, as measured by human evaluation Elo scores.

TTT-MLP preserves temporal consistency over scene changes and across angles.

Gated DeltaNet lacks temporal consistency across different angles of Tom.

TTT-MLP preserves temporal consistency over scene changes, producing smooth actions.



Sliding-window attention alters the kitchen environment and duplicates Jerry stealing the pie.

“Data” work

PufferLib 2.0: Reinforcement Learning at 1M steps/s

Joseph Suarez

Keywords: PufferLib, Reinforcement Learning, Library, Tools

https://puffer.ai/

Summary

PufferLib is an open-source reinforcement learning project built around efficient and broadly compatible simulation. Our first-party suite of 12 environments each run at 1M steps/second. For existing environments, PufferLib provides one-line wrappers that eliminate common compatibility problems and fast vectorization to accelerate training. With PufferLib, you can use familiar libraries like CleanRL and SB3 to scale from classic benchmarks like Atari and Procgen to complex simulators like NetHack and Neural MMO 3. Code, documentation, demos, and less formal blog coverage are available at puffer.ai.

1 Background and Introduction

Reinforcement learning research is slow and cumbersome. Atari games have been the most widely used benchmark since the introduction of the Arcade Learning Environment (Bellemare et al., 2012) in 2012. These are single-agent tasks that run at a few thousand steps per second on a modern CPU core. Standard learning libraries like CleanRL and SB3 then introduce old, unoptimized multiprocessing and expensive Python environment wrappers. The result is training that runs at hundreds to thousands of steps per second, leaving modern GPUs at 5-20% utilization. This is particularly limiting for RL research given the common sensitivity to hyperparameters. Accurately comparing methods often requires hundreds or thousands of experiments. This is impractical without large-scale industry resources.

Robust Autonomy Emerges from Self-Play

Marco Cusumano-Towner [*1] **David Hafner** [*1] **Alex Hertzberg** [*1] **Brody Huval** [*1] **Aleksei Petrenko** [*1]
Eugene Vinitsky [*1] **Erik Wijmans** [*1] **Taylor Killian** [1] **Stuart Bowers** [1] **Ozan Sener** [1]
Philipp Krähenbühl [1] **Vladlen Koltun** [1]

Abstract

Self-play has powered breakthroughs in two-player and multi-player games. Here we show that self-play is a surprisingly effective strategy in another domain. We show that robust and naturalistic driving emerges entirely from self-play in simulation at unprecedented scale – 1.6 billion km of driving. This is enabled by GIGAFLOW, a batched simulator that can synthesize and train on 42 years of subjective driving experience per hour on a single 8-GPU node. The resulting policy achieves state-of-the-art performance on three independent autonomous driving benchmarks. The policy outperforms the prior state of the art when tested on recorded real-world scenarios, amidst human drivers, without ever seeing human data during training. The policy is realistic when assessed against human references and achieves unprecedented robustness, averaging 17.5 years of continuous driving between incidents in simulation.

experiments (Feng et al., 2023; Zhang et al., 2023). This discovery is enabled by GIGAFLOW, a batched simulator architected from the ground up for self-play reinforcement learning on a massive scale. GIGAFLOW is capable of simulating and learning from 4.4 billion state transitions (7.2 million km of driving, or 42 years of continuous driving experience) per hour on a single 8-GPU node. It simulates urban environments with up to 150 densely interacting traffic participants 360 000 times faster than real time at a cost of under $5 per million km driven (based on public cloud rates). A full training run simulates over one trillion state transitions, 1.6 billion km driven, or 9500 years of subjective driving experience, and completes in under 10 days one 8-GPU node.

We use GIGAFLOW to train a parameterized family of driving policies. The parameters specify the type of traffic participant controlled by the policy (passenger vehicle, large truck, bicyclist, or even a pedestrian) and the driving style (e.g.,, aggressive vs. cautious). These parameters can be modified at test time with no additional training (Dosovitskiy & Koltun, 2017), such that a single trained policy can

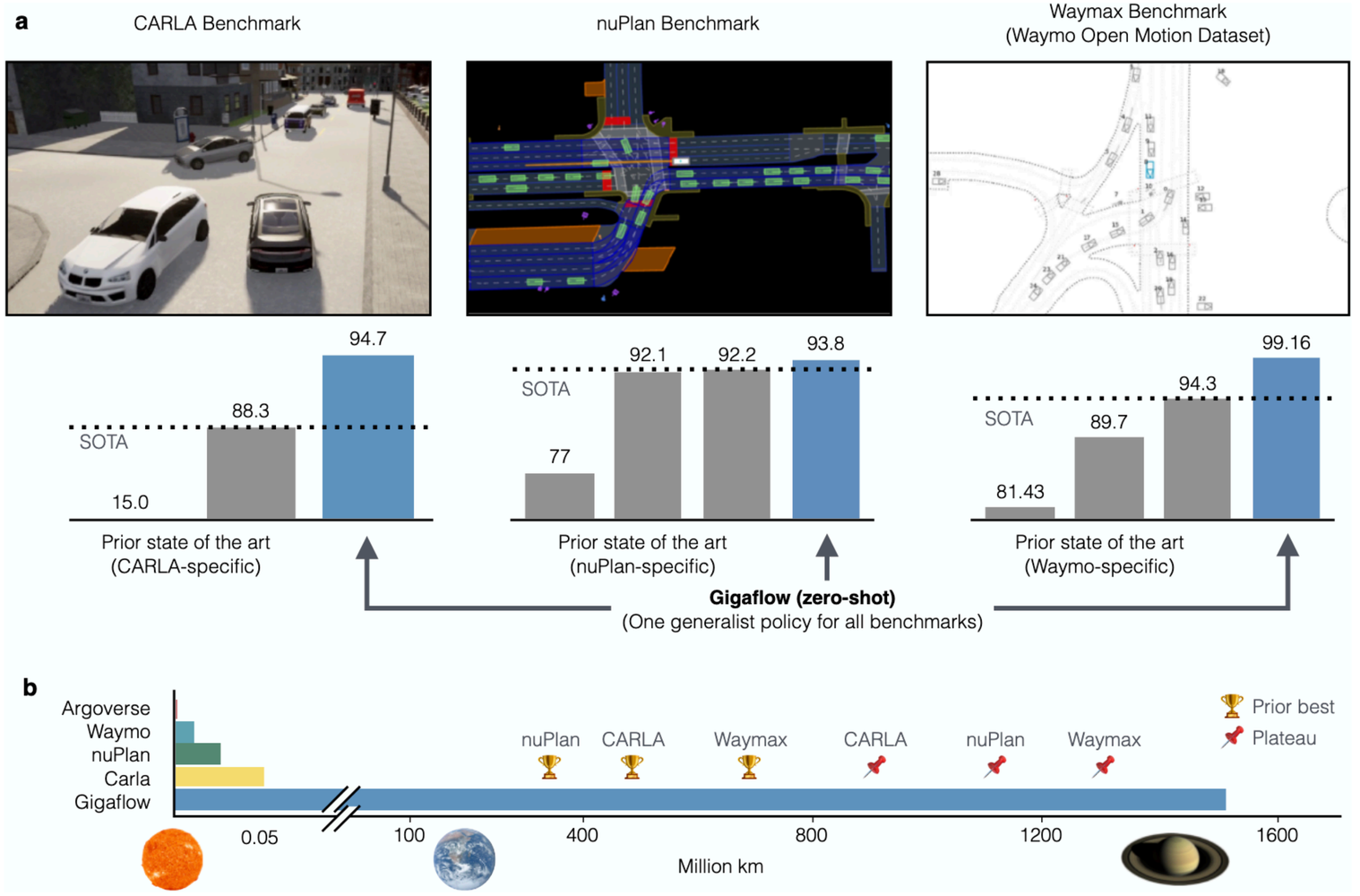
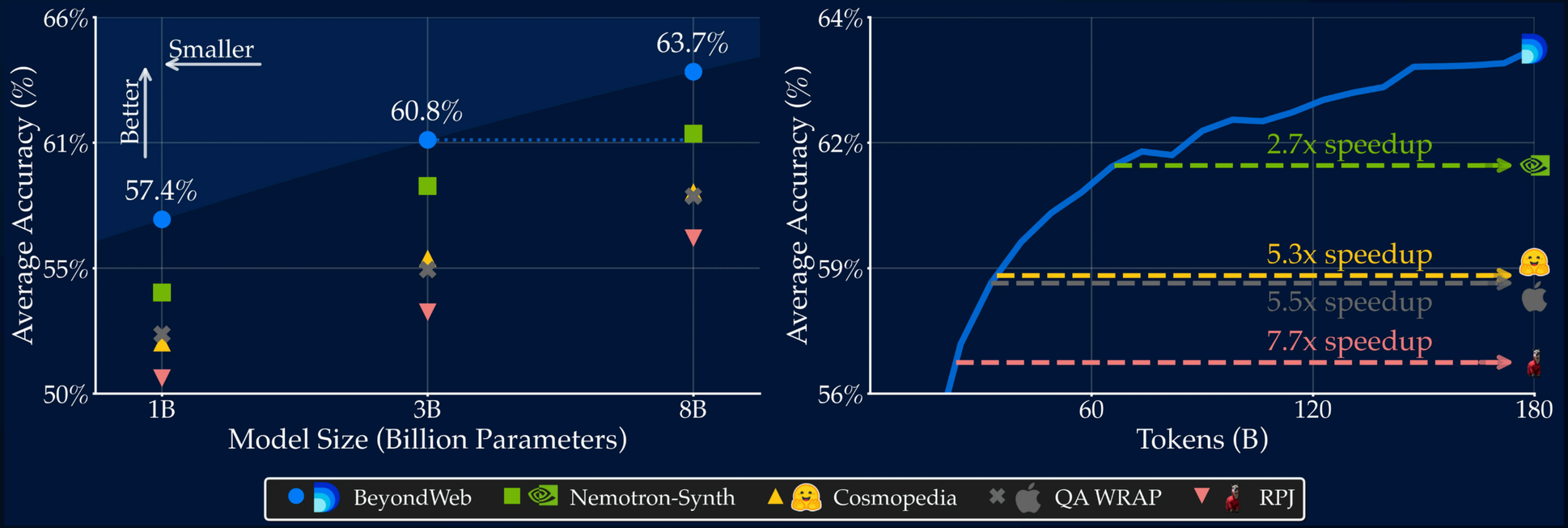


Figure 1: **Self-play reinforcement learning yields a generalist policy**. **a**, A single GIGAFLOW agent outperforms the best dataset-specific specialists across leading benchmarks. GIGAFLOW is evaluated zero-shot without training on a single benchmark, while dataset-specific specialists train on benchmark-specific datasets. Each benchmark includes different maps, scenarios, and evaluation metrics. **b**, GIGAFLOW enables cost-effective training of policies via self-play on a massive scale. Our largest policies drive over 1.6 billion km during training, more than the distance from the Sun to Saturn and orders of magnitude farther than prior datasets or simulations. At this scale, self-play yields a generalist policy. Dashed lines indicate points at which the performance of our single generalist policy passes the prior state of the art on each benchmark ('prior best') and points at which the performance on each benchmark plateaus ('plateau').

DatologyAI ▪ Aug 18 ▪ 59 min read

BeyondWeb: Lessons from Scaling Synthetic Data for Trillion-scale Pretraining



Links:

Roughly chronological

- https://how.rl.works & https://isattentionallyouneed.com – fun
- http://incompleteideas.net/IncIdeas/BitterLesson.html – the bitter lesson itself
- youtube.com/@asianometry – economics/ai chips/technology/history, kind-a 3b1b like
- https://www.yitay.net/blog/model-architecture-blogpost-encoders-prefixlm-denoising
  blog that touches on transformer architecture ideas from the following talk
- Stanford Cs25: V4 I Jason Wei & Hyung Won Chung of OpenAI – ideas about data and architecture progress
  maybe even too much bitter-lesson pilled
- Linear Attention and Beyond – Songling Yang talk about all sorts of sub-quadratic models