

Jasmine: A Simple, Performant and Scalable JAX-based World Modeling Codebase

Mihir Mahajan*1,2, Alfred Nguyen*1,2, Franz Srambical*1,2

Stefan Bauer²

*Contributed equally

¹p(doom), ²TUM

While world models are increasingly positioned as a pathway to overcoming data scarcity in domains such as robotics, open training infrastructure for world modeling remains nascent. We introduce Jasmine, a performant JAX-based world modeling codebase that scales from single hosts to hundreds of accelerators with minimal code changes. Jasmine achieves an order-of-magnitude faster reproduction of the CoinRun case study compared to prior open implementations, enabled by performance optimizations across data loading, training and checkpointing. The codebase guarantees fully reproducible training and supports diverse sharding configurations. By pairing Jasmine with curated large-scale datasets, we establish infrastructure for rigorous benchmarking pipelines across model families and architectural ablations.

Keywords: World Modeling, JAX, Reinforcement Learning

1. Introduction

Over the past decades, the field of deep learning has increasingly been shaped by methods that leverage vast data troves (Chowdhery et al., 2023; Deng et al., 2009; Fedus et al., 2022; Jozefowicz et al., 2016; Radford et al., 2018, 2019; Raffel et al., 2020), and paradigms that unlock new ones (Berner et al., 2019; Christiano et al., 2017; Guo et al., 2025a; Radford et al., 2021; Silver et al., 2016; Srambical and Mahajan, 2025). Internet-scale pre-training, preference modeling, and reinforcement learning using verification signals offer a compelling pathway for language models to attain human-level performance (Lin and Cheng, 2025; Luong and Lockhart, 2025), yet data is increasingly bottlenecking progress from spiky towards general intelligence. While some domains can leverage user feedback from deployed products for iterative model improvement (Cursor, 2025), domains like robotics cannot afford such a privilege.

One paradigm proposed by the research community to overcome the data scarcity in those domains is that of world models (Ha and Schmidhuber, 2018). World models can aid frontier model development in numerous ways, but one particularly ambitious goal of the community is to train a world model to act as a simulation of the real world (Agarwal et al., 2025; Ball et al., 2025; Bruce et al., 2024; Parker-Holder et al., 2022), in order to train agents in that simulation (Hafner et al., 2025b), via an adaptive curriculum (Parker-Holder et al., 2022), or otherwise. This regime requires the compounding error of the world model to be orders of magnitude smaller than when solely used for short-term look-ahead. The feasibility of such a world model in its truest sense is entirely understudied. Jasmine provides the foundational infrastructure for future empirical investigation of how compute and data requirements scale with environment complexity for downstream agent training.

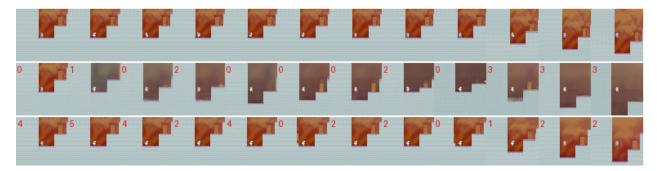


Figure 1 | Autoregressive sampling of Jafar (Willi et al., 2024) (middle row) and Jasmine (bottom row) on the CoinRun case study with four conditioning frames (conditioning frames not shown). The top row shows the ground-truth sequence.

2. Jasmine

Our contributions in this work are threefold: i) we introduce Jasmine, a highly optimized and scalable JAX-based codebase for world modeling, which we use to reproduce Genie's CoinRun case study (Bruce et al., 2024; Cobbe et al., 2020) an order of magnitude faster than prior work (Willi et al., 2024). This speedup is the result of infrastructure optimizations, including a fully reproducible, scalable training and data pipeline built on the JAX (Bradbury et al., 2018) ecosystem. ii) We find that a critical modification to the original Genie architecture, prepending latent actions instead of adding them to video embeddings, is required for the world model to yield generations that faithfully reproduce the CoinRun environment. iii) Finally, we openly release the Jasmine codebase, along with pretrained checkpoints, curated datasets, model inspection notebooks, and a dataset of dense IDE interactions captured during Jasmine's development, providing the first openly published dataset of months-long software engineering.

Jasmine implements the Genie (Bruce et al., 2024) architecture, enabling training of interactive environments from unlabeled videos. The architecture includes a video tokenizer, which encodes videos into tokens, a latent action model (LAM) that extracts latent actions between video frames, and a dynamics model that predicts the tokens of the next frame based on the previous tokens and corresponding latent actions. At sampling time, the LAM is discarded and replaced by input from the user. All modules use an ST-Transformer (Ho et al., 2019) backbone which approximates full attention by performing intra-frame (spatial) followed by inter-frame (temporal) attention, thus reducing the attention sequence length. The tokenizer uses a VQ-VAE (Van Den Oord et al., 2017) to encode image patches using reconstruction, vector-quantization, and commitment losses. To train an action-conditioned video-generation model from unlabeled videos, Genie learns latent actions (Schmidt and Jiang, 2024). Like the tokenizer, the LAM uses a VQ-VAE, with its codebook representing the latent actions. The model learns to distill information from future frames into this bottlenecked codebook: Frames $x_{0:t}$ are encoded, producing latent actions $a_{0:t}$, which the decoder receives along with past frames $x_{0:t-1}$ to predict the next frame x_t . A temporal causal mask allows the entire sequence to be processed in a single forward pass. The dynamics model is a decoder-only transformer that predicts future frames conditioned on past frames and corresponding latent actions. Genie uses MaskGIT (Chang et al., 2022), which masks input-tokens at training time, similar in spirit to BERT (Devlin et al., 2019). Unlike MaskGIT, Genie masks with probability $p \sim U(0.5, 1)$ (refer to Appendix **F** for details about extending MaskGIT to videos).

Building upon prior work (Willi et al., 2024) that openly published a reimplementation of the Genie (Bruce et al., 2024) architecture, we release a highly optimized JAX-based world modeling codebase amenable to scale. Jasmine implements a range of baselines, including MaskGIT-based (Chang et al.,

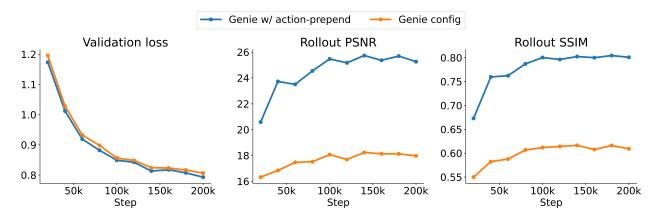


Figure 2 | Validation metrics of the CoinRun case study (patch size 4). While the loss (left) is similar between the default Genie configuration and our minimal modification, rollout metrics (middle and right, refer to Section G) differ substantially.

2022), fully causal (Srambical, 2024), and diffusion-based approaches (Appendix C). The codebase depends solely on battle-tested libraries from the Google ecosystem (JAX, NNX, Grain (Ritter et al., 2023), Orbax, Optax, Treescope (Johnson, 2024), ArrayRecord (Google, 2024)), and scales from single hosts to hundreds of accelerators using XLA. Jasmine supports complex sharding configurations in a few lines of code through Shardy (OpenXLA Project, 2025). It provides asynchronous distributed checkpointing with configurable policies, process-parallel dataloading, and checkpointing of model, optimizer, and data loader states. Training runs are bitwise deterministic, yielding identical loss curves under identical seeds (Appendix D). To enable efficient large-scale experimentation, Jasmine integrates mixed-precision, FlashAttention via cuDNN SDPA (NVIDIA Corporation, 2025), activation checkpointing, host memory offloading, and index-shuffling during data loading. The codebase follows the shape suffix convention of Shazeer (2024), aiming to provide a didactic implementation of modern world modeling architectures.

We run the reproducible case study described in Bruce et al. (2024) by generating a dataset containing 50M transitions of CoinRun, an environment of the Procgen benchmark (Cobbe et al., 2020) (Appendix A). In contrast to Bruce et al. (2024) we find that strict adoption of the architecture and hyperparameters described in their case study leads to deteriorating autoregressive generations in both Jasmine and Jafar (Willi et al., 2024) (Figure 4, middle row). However, a minimal modification of the case study setting, namely prepending latent actions instead of adding them to the video embeddings, yields autoregressive generations that faithfully simulate the CoinRun environment (Figure 4, bottom row and Figure 2). We hypothesize that this discrepancy between Bruce et al. (2024) and our work stems from an ambiguity in extending MaskGIT to videos (refer to Appendix F for further discussion).

Beyond openly publishing¹ the Jasmine codebase, we recorded every keystroke during Jasmine's development using *crowd-code* (Srambical and Mahajan, 2025), a VS Code/Cursor extension that enables crowd-sourcing dense IDE interaction data. To our knowledge, this represents one of the first open datasets capturing the full temporal scale of months-long software development, thus laying groundwork for future work in behaviour cloning, goal-conditioning, and verification signal mining.

¹https://github.com/p-doom/jasmine

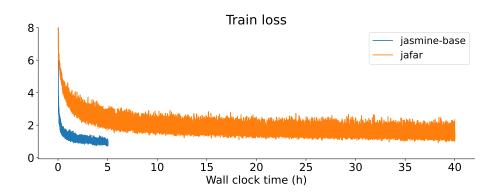


Figure 3 | An order of magnitude faster convergence in wall-clock time in Jasmine (blue) compared to Jafar (Willi et al., 2024) (orange). We report the train loss since Jafar does not collect validation metrics. Refer to Appendix B for Jasmine's validation metrics. Jasmine's lower variance stems from a subtle refinement in its batched masking logic (Appendix H).

3. Experiments

We evaluate the performance of Jasmine and analyze the impact of its core components through rigorous ablations. Using Jasmine, we reproduce the CoinRun case study at a patch size of 16 in under nine hours on a single GPU, compared to over 100 hours reported in prior work (Willi et al., 2024) under the same setting (Figure 3). We present ablations identifying the factors responsible for this speedup in Table 2. Section B further reports results from architectural modifications including replacing the latent action model with ground-truth actions, ablating co-training, adopting fully causal and diffusion baselines, and setting the feedforward expansion factor to four.

Architectural optimizations We adapt Genie's architectural choices by integrating best practices from the language modeling community. Specifically, we use a feedforward expansion factor of four relative to the model dimension, following common practice in large-scale language modeling (Brown et al., 2020; Radford et al., 2019; Raffel et al., 2020). We simultaneously reduce network depth, resulting in lower overall parameter count than the Genie defaults, thus achieving higher throughput (Table 1) while maintaing competitive performance (Figure 5). We employ the warmup-stable-decay (WSD) learning rate schedule (Mahajan et al., 2018; Zhai et al., 2022), which allows flexible training durations by resuming from a checkpoint prior to the decay phase. Unlike (Bruce et al., 2024), we warm up from and decay the learning rate to zero, in line with established best practices (Zhai et al., 2022). We further compare co-training LAM and dynamics model (as done in Bruce et al. (2024)) with pre-training the LAM (as done in Willi et al. (2024)), embedding ground-truth actions instead of using the latent action model (Appendix B), and replacing MaskGIT with fully causal and diffusion baselines. Co-training, pre-training the LAM, and using ground-truth actions are all competitive (Figure 5), while the fully causal baseline underperforms in the 200k steps training regime (Figure 6). However, our results indicate that the fully causal baseline in particular may benefit from longer training. Diffusion-forcing (Chen et al., 2024) outperforms MaskGIT, even when using identical per-frame sampling step counts and untuned hyperparameters (Figure 7, Appendix C).

Infrastructure optimizations A substantial portion of our speedup compared to Willi et al. (2024) arises from our data loader design (Tables 4 and 5). We use Grain for data loading with prefetching enabled and preprocess datasets into ArrayRecords (Google, 2024), a file format optimized for random

	Throughput (bs=36)	Throughput (bs=2048)
Jasmine-base	1.00x	1.00x
1x feedforward expansion	0.93x	0.79x

Table 1 | Training throughput with a feedforward expansion factor of one, relative to Jasmine-base. We double the number of layers compared to Jasmine-base to roughly match the parameter count (refer to Genie's default configuration in 6).

access indexing. The chosen chunking strategy significantly affects throughput, and we describe our configuration in Appendix B. Jasmine further leverages FlashAttention (Dao et al., 2022) via cuDNN SDPA (NVIDIA Corporation, 2025) and mixed precision training with bfloat16. We report throughputs when ablating mixed precision, FlashAttention and Grain in Table 2.

4. Related Work and Discussion

Although research on world models with its inception decades ago (Sutton, 1991) has matured over the years (Alonso et al., 2024; Ha and Schmidhuber, 2018; Hafner et al., 2019, 2020a,b, 2025a), they have only been scaled up recently (Agarwal et al., 2025; Ball et al., 2025; Bruce et al., 2024; Decart et al., 2024; Guo et al., 2025b; Hafner et al., 2025b; Hu et al., 2023; Li et al., 2025; Parker-Holder et al., 2024; Pearce et al., 2025; Seid and Hojel, 2024; Valevski et al., 2025). While the open training ecosystem in language modeling provides mature solutions for large-scale language pretraining (Shoeybi et al., 2019; Witten and MaxText Authors, 2024), open training infrastructure for world modeling is still nascent (Castricato et al., 2025; Savov et al., 2025). Closest to our work is Willi et al. (2024), an open-source reproduction of Genie (Bruce et al., 2024), which we build upon and significantly extend.

With Jasmine we make progress towards democratizing world modeling research. Alongside the codebase, we openly release checkpoints and datasets for CoinRun, Atari and Doom, as well as dense IDE interaction data collected over months of research engineering². While Jasmine greatly accelerates wall-clock convergence compared to prior work, it has yet to match throughput efficiencies of frontier language model implementations.

Acknowledgments

We thank Matthew T. Jackson, Andrea Dittadi and Diego Marti Monso for useful discussions as well as the Jafar authors for openly publishing their repository. The authors gratefully acknowledge the computing time provided on the high-performance computer HoreKa by the National High-Performance Computing Center at KIT (NHR@KIT). This center is jointly supported by the Federal Ministry of Education and Research and the Ministry of Science, Research and the Arts of Baden-Württemberg, as part of the National High-Performance Computing (NHR) joint funding program (https://www.nhr-verein.de/en/our-partners). HoreKa is partly funded by the German Research Foundation (DFG).

²Datasets published under the CCO license at https://huggingface.co/datasets/p-doom

References

- N. Agarwal, A. Ali, M. Bala, Y. Balaji, E. Barker, T. Cai, P. Chattopadhyay, Y. Chen, Y. Cui, Y. Ding, et al. Cosmos world foundation model platform for physical ai. *arXiv preprint arXiv:2501.03575*, 2025.
- E. Alonso, A. Jelley, V. Micheli, A. Kanervisto, A. J. Storkey, T. Pearce, and F. Fleuret. Diffusion for world modeling: Visual details matter in atari. In *Advances in neural information processing systems*, volume 37, 2024.
- P. J. Ball, J. Bauer, F. Belletti, B. Brownfield, A. Ephrat, S. Fruchter, A. Gupta, K. Holsheimer, A. Holynski, J. Hron, C. Kaplanis, M. Limont, M. McGill, Y. Oliveira, J. Parker-Holder, F. Perbet, G. Scully, J. Shar, S. Spencer, O. Tov, R. Villegas, E. Wang, J. Yung, C. Baetu, J. Berbel, D. Bridson, J. Bruce, G. Buttimore, S. Chakera, B. Chandra, P. Collins, A. Cullum, B. Damoc, V. Dasagi, M. Gazeau, C. Gbadamosi, W. Han, E. Hirst, A. Kachra, L. Kerley, K. Kjems, E. Knoepfel, V. Koriakin, J. Lo, C. Lu, Z. Mehring, A. Moufarek, H. Nandwani, V. Oliveira, F. Pardo, J. Park, A. Pierson, B. Poole, H. Ran, T. Salimans, M. Sanchez, I. Saprykin, A. Shen, S. Sidhwani, D. Smith, J. Stanton, H. Tomlinson, D. Vijaykumar, L. Wang, P. Wingfield, N. Wong, K. Xu, C. Yew, N. Young, V. Zubov, D. Eck, D. Erhan, K. Kavukcuoglu, D. Hassabis, Z. Gharamani, R. Hadsell, A. van den Oord, I. Mosseri, A. Bolton, S. Singh, and T. Rocktäschel. Genie 3: A new frontier for world models. 2025. URL https://deepmind.google/discover/blog/genie-3-a-new-frontier-for-world-models/.
- C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv* preprint *arXiv*:1912.06680, 2019.
- J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/jax-ml/jax.
- T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. In *Advances in neural information processing systems*, volume 33, 2020.
- J. Bruce, M. D. Dennis, A. Edwards, J. Parker-Holder, Y. Shi, E. Hughes, M. Lai, A. Mavalankar, R. Steigerwald, C. Apps, Y. Aytar, S. M. E. Bechtle, F. Behbahani, S. C. Chan, N. Heess, L. Gonzalez, S. Osindero, S. Ozair, S. Reed, J. Zhang, K. Zolna, J. Clune, N. de Freitas, S. Singh, and T. Rocktäschel. Genie: Generative interactive environments. In *Proceedings of the 41st International Conference on Machine Learning*, 2024.
- L. Castricato, S. Matiana, A. Lapp, and S. BuGhanem. owl-wms: Basic world models, 2025. URL https://github.com/Wayfarer-Labs/owl-wms.
- H. Chang, H. Zhang, L. Jiang, C. Liu, and W. T. Freeman. Maskgit: Masked generative image transformer. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022.
- B. Chen, D. Martí Monsó, Y. Du, M. Simchowitz, R. Tedrake, and V. Sitzmann. Diffusion forcing: Next-token prediction meets full-sequence diffusion. In *Advances in neural information processing systems*, volume 37, 2024.
- A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, et al. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240), 2023.

- P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei. Deep reinforcement learning from human preferences. In *Advances in neural information processing systems*, volume 30, 2017.
- K. Cobbe, C. Hesse, J. Hilton, and J. Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *Proceedings of the 37th International Conference on Machine Learning*, 2020.
- Cursor. A new tab model, 2025. URL https://cursor.com/blog/tab-update.
- T. Dao, D. Fu, S. Ermon, A. Rudra, and C. Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. In *Advances in neural information processing systems*, volume 35, 2022.
- E. Decart, Q. McIntyre, S. Campbell, X. Chen, and R. Wachen. Oasis: A universe in a transformer, 2024. URL https://oasis-model.github.io.
- J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2009.
- J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, 2019.
- W. Fedus, B. Zoph, and N. Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120), 2022.
- K. Frans, D. Hafner, S. Levine, and P. Abbeel. One step diffusion via shortcut models. *arXiv preprint arXiv:2410.12557*, 2024.
- Google. Arrayrecord: A file format for efficient io, 2024. URL https://github.com/google/array_record.
- D. Guo, D. Yang, H. Zhang, J. Song, P. Wang, Q. Zhu, R. Xu, R. Zhang, S. Ma, X. Bi, et al. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081), 2025a.
- J. Guo, Y. Ye, T. He, H. Wu, Y. Jiang, T. Pearce, and J. Bian. Mineworld: a real-time and open-source interactive world model on minecraft. *arXiv* preprint arXiv:2504.08388, 2025b.
- D. Ha and J. Schmidhuber. Recurrent world models facilitate policy evolution. In *Advances in neural information processing systems*, volume 31, 2018.
- D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson. Learning latent dynamics for planning from pixels. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.
- D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi. Dream to control: Learning behaviors by latent imagination. In *The Eighth International Conference on Learning Representations*, 2020a.
- D. Hafner, T. Lillicrap, M. Norouzi, and J. Ba. Mastering atari with discrete world models. *arXiv* preprint arXiv:2010.02193, 2020b.
- D. Hafner, J. Pasukonis, J. Ba, and T. Lillicrap. Mastering diverse control tasks through world models. *Nature*, 2025a.

- D. Hafner, W. Yan, and T. Lillicrap. Training agents inside of scalable world models. *arXiv preprint arXiv:2509.24527*, 2025b.
- K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022.
- J. Ho, N. Kalchbrenner, D. Weissenborn, and T. Salimans. Axial attention in multidimensional transformers. *arXiv preprint arXiv:1912.12180*, 2019.
- A. Hu, L. Russell, H. Yeo, Z. Murez, G. Fedoseev, A. Kendall, J. Shotton, and G. Corrado. Gaia-1: A generative world model for autonomous driving. *arXiv preprint arXiv:2309.17080*, 2023.
- D. D. Johnson. Penzai + Treescope: A toolkit for interpreting, visualizing, and editing models as data. *ICML 2024 Workshop on Mechanistic Interpretability*, 2024.
- R. Jozefowicz, O. Vinyals, M. Schuster, N. Shazeer, and Y. Wu. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*, 2016.
- J. Li, J. Tang, Z. Xu, L. Wu, Y. Zhou, S. Shao, T. Yu, Z. Cao, and Q. Lu. Hunyuan-gamecraft: High-dynamic interactive game video generation with hybrid history condition. *arXiv* preprint *arXiv*:2506.17201, 2025.
- H. M. Lin and H.-T. Cheng. Gemini achieves gold-level performance at the international collegiate programming contest world finals, 2025. URL https://deepmind.google/discover/blog/gemini-achieves-gold-level-performance-at-the-international-collegiate-programming-contest-world-finals/.
- T. Luong and E. Lockhart. Advanced version of gemini with deep think officially achieves gold-medal standard at the international mathematical olympiad, 2025. URL https://deepmind.google/discover/blog/advanced-version-of-gemini-with-deep-think-officially-achieves-gold-medal-standard-at-the-international-mathematical-olympiad/.
- D. Mahajan, R. Girshick, V. Ramanathan, K. He, M. Paluri, Y. Li, A. Bharambe, and L. Van Der Maaten. Exploring the limits of weakly supervised pretraining. In *Proceedings of the European conference on computer vision*, 2018.
- NVIDIA Corporation. cudnn frontend api: Scaled dot product attention fp16/bf16 forward, 2025. URL https://docs.nvidia.com/deeplearning/cudnn/frontend/latest/operations/Attention.html.
- OpenXLA Project. Shardy: an mlir-based tensor partitioning system, 2025. URL https://github.com/openxla/shardy.
- J. Parker-Holder, M. Jiang, M. Dennis, M. Samvelyan, J. Foerster, E. Grefenstette, and T. Rocktäschel. Evolving curricula with regret-based environment design. In *Proceedings of the 39th International Conference on Machine Learning*, 2022.
- J. Parker-Holder, P. Ball, J. Bruce, V. Dasagi, K. Holsheimer, C. Kaplanis, A. Moufarek, G. Scully, J. Shar, J. Shi, S. Spencer, J. Yung, M. Dennis, S. Kenjeyev, S. Long, V. Mnih, H. Chan, M. Gazeau, B. Li, F. Pardo, L. Wang, L. Zhang, F. Besse, T. Harley, A. Mitenkova, J. Wang, J. Clune, D. Hassabis, R. Hadsell, A. Bolton, S. Singh, and T. Rocktäschel. Genie 2: A large-scale foundation world model. 2024. URL https://deepmind.google/discover/blog/genie-2-a-large-scale-foundation-world-model/.

- T. Pearce, T. Rashid, D. Bignell, R. Georgescu, S. Devlin, and K. Hofmann. Scaling laws for pre-training agents and world models. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025.
- W. Peebles and S. Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF* conference on computer vision and pattern recognition, 2023.
- A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, et al. Improving language understanding by generative pre-training. *OpenAI blog*, 2018.
- A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 2019.
- A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, et al. Learning transferable visual models from natural language supervision. In *Proceedings of the 38th International Conference on Machine Learning*, 2021.
- C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140), 2020.
- M. Ritter, I. Indyk, A. Singh, A. Audibert, A. Seelam, C. Hanes, E. Lau, J. Olesiak, J. Kang, and X. Wu. Grain feeding jax models, 2023. URL http://github.com/google/grain.
- J. K. Salmon, M. A. Moraes, R. O. Dror, and D. E. Shaw. Parallel random numbers: as easy as 1, 2, 3. In *Proceedings of 2011 international conference for high performance computing, networking, storage and analysis*, 2011.
- N. Savov, N. Kazemi, M. Mahdi, D. P. Paudel, X. Wang, and L. Van Gool. Exploration-driven generative interactive environments. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2025.
- D. Schmidt and M. Jiang. Learning to act without actions. In *The Twelfth International Conference on Learning Representations*, 2024.
- R. Seid and A. Hojel. Lucid v1: Real-time latent world models, 2024. URL https://www.lucid.ai/notes/lucid-v1.
- N. Shazeer. Shape suffixes, 2024. URL https://medium.com/@NoamShazeer/shape-suffixes-good-coding-style-f836e72e24fd.
- M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv* preprint arXiv:1909.08053, 2019.
- D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587), 2016.
- F. Srambical. Going beyond the causal mask in language modeling. *p*(*doom*) *blog*, 2024. URL https://pdoom.org/blog.html.
- F. Srambical and M. Mahajan. Crowd-sourcing a dataset to make agents code like humans. *p*(*doom*) *blog*, 2025. URL https://pdoom.org/blog.html.

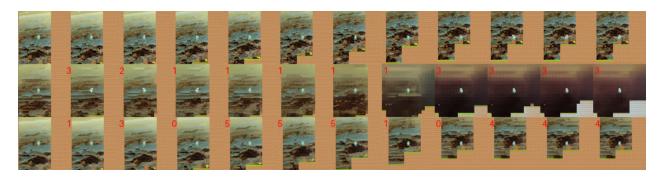


Figure 4 | Autoregressive sampling of Jasmine when adding (middle row) and prepending actions (bottom row) on the CoinRun case study with four conditioning frames (conditioning frames not shown). The top row shows the ground-truth sequence.

- R. S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM Sigart Bulletin*, 2(4), 1991.
- D. Valevski, Y. Leviathan, M. Arar, and S. Fruchter. Diffusion models are real-time game engines. In *The Thirteenth International Conference on Learning Representations*, 2025.
- A. Van Den Oord, O. Vinyals, and K. Kavukcuoglu. Neural discrete representation learning. In *Advances in neural information processing systems*, volume 30, 2017.
- T. Willi, M. T. Jackson, and J. N. Foerster. Jafar: An open-source genie reimplementation in jax. In *First Workshop on Controllable Video Generation @ ICML 2024*, 2024.
- R. Witten and MaxText Authors. MaxText: A simple, performant and scalable Jax LLM, 2024. URL https://github.com/google/maxtext.
- X. Zhai, A. Kolesnikov, N. Houlsby, and L. Beyer. Scaling vision transformers. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022.
- R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2018.

A. Coinrun Case Study

For the CoinRun case study, we strictly adhere to the setting of Bruce et al. (2024) and train our models to unmask sequences of 16 frames with a resolution of 64x64 pixels per frame. To generate the dataset, we capture 50M observation frames and corresponding ground-truth actions during random agent rollouts and only use the ground-truth actions for a LAM ablation (Section B). Instead of sampling seeds from a fixed pool as described in Bruce et al. (2024), we initialize all episodes with a seed unique to the respective episode. Furthermore, we verify that our generated dataset contains no duplicate episodes and only find 7.46% duplicate frames. We confirm that the validation and test set are disjoint from the train set and publish our script for duplication detection along with the repository³. While train metrics are near-identical between Genie's configuration and our action-prepending modification, rollout quality differs significantly (Figure 2). We collect rollout metrics during training (Section G) that capture this discrepancy.

 $^{^3} https://github.com/p-doom/jasmine/blob/main/data/jasmine_data/detect_array_record_duplicates.py$

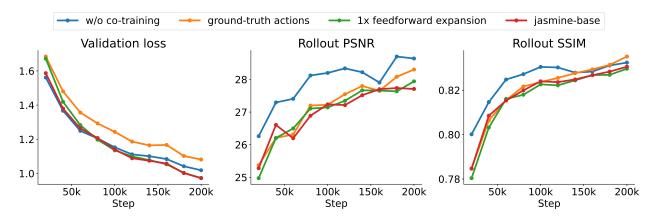


Figure 5 | Architectural ablations of Jasmine's base configuration (refer to Table 6) on CoinRun. We report loss (left) and rollout metrics (middle and right) of the dynamics model on a validation set.

B. Ablations

For our ablations, we reuse the CoinRun setting and ablate from Jasmine's base configuration, depicted in Table 6.

Co-training LAM and dynamics model Bruce et al. (2024) co-train the LAM and the dynamics model. However, their implementation remains unclear as the LAM is supervised on frames while the dynamics model is supervised on tokens. One approach to co-training is a combined loss function including stop-gradients that prevent gradients from flowing from the dynamics model to the LAM. However, such a combined loss formulation remains unmentioned in Bruce et al. (2024). Willi et al. (2024) instead train LAM and dynamics model sequentially, thus reducing memory footprint at the cost of longer total training time. For Jasmine's co-training implementation, we omit the LAM decoder entirely and allow gradients to flow from the dynamics model to the LAM.

Training with ground-truth actions We ablate the LAM (Figure 5) by training the dynamics model using ground-truth actions captured in the environment. We use an embedding layer to map action indices to action latents, which are then used as additional input to the dynamics model.

Throughput ablations We ablate core components of Jasmine's infrastructure in Table 2. Replacing Grain with the default data loader of Willi et al. (2024) reduces throughput by an order of magnitude. In Jasmine's base CoinRun configuration (sub-100M parameter model and a maximum attention sequence length of 16) the XLA compiler dispatches higher-throughput CUDA kernels than FlashAttention. While FlashAttention only outperforms XLA-compiled kernels at large model sizes and sequence lengths (Table 3), we enable it by default to reduce accelerator memory usage. At small batch sizes, the compiled train loop of our configuration achieves higher throughput using full precision. We attribute this to XLA largely operating based on heuristics, and posit that writing optimized Pallas kernels for key operations in the model forward pass will result in mixed precision outperforming full precision in throughput, even at small batch sizes.

The ArrayRecord file format allows storing a configurable amount of records per file. In our case, each record corresponds to a sequence of frames and actions. We find that the chosen format significantly affects throughput (Tables 4 and 5). Based on preliminary experiments, we preprocess the dataset to have 100 records per ArrayRecord file with 160 frames per record.

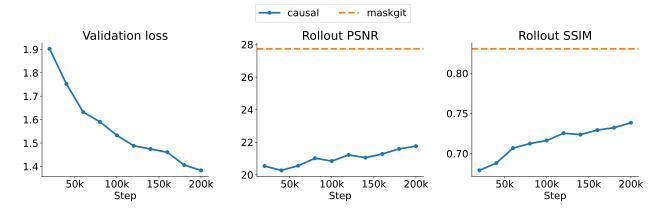


Figure 6 | Loss (left) and rollout metrics (middle and right) of the fully causal baseline. We depict the final performance of our MaskGIT implementation (Jasmine's default configuration) for the rollout metrics. The loss indicates that the causal baseline might benefit from longer training and separately tuned hyperparameters. The losses between the two architectures are not comparable, hence we omit the MaskGIT loss.

	Throughput (bs=36)	Throughput (bs=2048)
Jasmine-base	1.00x	1.00x
w/o grain data loader	0.25x	0.11x
w/o flash attention	1.15x	1.04x
w/o mixed precision	1.18x	0.71x

Table 2 | Training throughput of infrastructure ablations, relative to Jasmine-base. We report the throughput at Genie's default batch size (36), as well as at the batch size resulting in the highest throughput (2048) on a single H100 with 80GB of accelerator memory.

	Throughput (frames/sec)	Throughput (relative)
w/ flash attention	36.15	1.00x
w/o flash attention	24.24	0.67x

Table 3 | Training throughput using a larger model (1B parameter) and spatial sequence length (1024). We decrease the patch size to two. In this regime, FlashAttention yields higher throughput than the XLA compiler.

# frames per record	# records per file	Throughput (frames/sec)	Throughput (relative)	
16	100	7,527.27	1.12x	
160 (Base)	100	6,709.09	1.00x	
1,600	100	3,752.73	0.56x	
16,000	100	3,720.00	0.55x	
160,000	100	3,785.45	0.56x	

Table 4 | Training throughput at Genie's default batch size (36) with different number of frames per record. Throughput decreases as the number of frames per record increases. We opt for 160 frames per record to be able to vary the sequence length.

# frames per record	# records per file	Throughput (frames/sec)	Throughput (relative)	
160	1	6,098.18	0.91x	
160	10	6,643.64	0.99x	
160 (Base)	100	6,709.09	1.00x	
160	1,000	6,480.00	0.97x	
160	10,000	6,763.64	1.01x	

Table 5 | Training throughput at Genie's default batch size (36) with different number of records per file.

C. Diffusion Baseline

We implement a diffusion baseline inspired by the Dreamer 4 architecture (Hafner et al., 2025b), combining a masked autoencoder (MAE, He et al. (2022)) tokenizer with an ST-DiT (Ho et al., 2019; Peebles and Xie, 2023) dynamics model trained under the diffusion-forcing objective (Chen et al., 2024). We leave implementing the shortcut objective (Frans et al., 2024) to future work.

Tokenizer Following Hafner et al. (2025b), we use a MAE to compress raw video frames into continuous latents. Our autoencoder implementation uses an ST-Transformer backbone and a latent bottleneck. Before passing the latents to the decoder, we apply the tanh activation to constrain them to the range (-1,1) for downstream dynamics model training. We uniformly sample per-frame masking probabilities $p_i \sim U(0,0.9)$. Unlike Hafner et al. (2025b), we omit the auxiliary LPIPS loss (Zhang et al., 2018) and directly train on pixel-level reconstructions using mean-squared error. We find the tokenizer hyperparameters from Table 6 to work well for MAE training as well.

Dynamics model We implement diffusion forcing (Chen et al., 2024), sampling an independent noise level per frame during training. Analogous to Hafner et al. (2025b) and Jasmine-base, we prepend latent actions and the embedded denoising step to the patch latents. Following Hafner et al. (2025b), we use x-prediction⁴ and employ a ramp loss. During inference, frame-wise latents are autoregressively generated with 25 denoising steps per frame, while past input latents are slightly corrupted using a noise level of 0.1. We adopt the hyperparameters of Jasmine-base, and only change the learning rate to 1e-4 following Peebles and Xie (2023).

D. Bitwise Determinism

On TPUs, bitwise determinism is guaranteed by Jasmine via proper usage of JAX's implementation of parallel random number generation via Threefry counters (Salmon et al., 2011). On GPUs however, an additional XLA flag (xla_gpu_deterministic_ops=true) is needed in certain cases to guarantee identical training curves.

⁴In the diffusion literature, x-prediction often refers to supervision in latent-space rather than pixel-space. We follow that nomenclature but believe that the term z-prediction is a more accurate description.

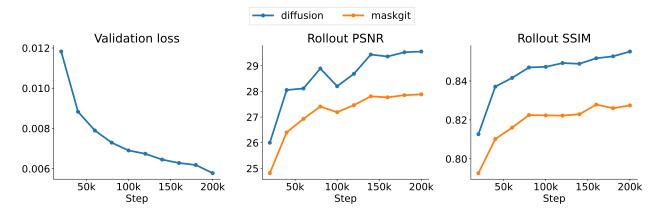


Figure 7 | Loss (left) and rollout metrics (middle and right) of the diffusion baseline. We omit the MaskGIT loss as the losses are not comparable between the two architectures.

E. Hyperparameter Configurations

We mention four distinct training configurations. We present hyperparameters and training settings of each configuration in Table 6 and briefly describe them:

- **Genie**: The hyperparameter configuration of Appendix F of Bruce et al. (2024), but with a patch size of 16 (for easier comparison against Jafar). We use this configuration for the CoinRun case study (Figure 4, middle row; refer to Section A).
- Genie w/ prepend: As detailed in Section 2, we found a minimal modification to the Genie configuration to be necessary to yield generations faithful to the CoinRun environment (Figure 1, bottom row and Figure 4, bottom row). This setting is identical to our Genie configuration, with the exception that we prepend latent actions to the video embeddings instead of adding them.
- Jafar: The hyperparameters used by Jafar (Willi et al., 2024) for their CoinRun case study. This setting is identical to Bruce et al. (2024), but Willi et al. (2024) use a patch size of 16 for faster training. Unlike Bruce et al. (2024), they pre-train the LAM as opposed to co-training the LAM with the dynamics model. We use this configuration for the Jafar baseline runs (Figure 1, middle row and Figure 3). We solely run this configuration with the Jafar repository.
- Jasmine-base: We define a base configuration for Jasmine that represents a trade-off between training speed, modeling quality and simplicity, integrating best practices from the language modeling literature. This is Jasmine's configuration in the wall-clock convergence comparison (Figure 3) between Jasmine and Jafar, as well as our architectural and infrastructure ablations (Figures 5, 10 and Tables 1, 2, 3, 4, 5).

F. Extending MaskGIT to Videos

MaskGIT (Chang et al., 2022) is defined on images and there are multiple ways to extend it to videos. We follow Willi et al. (2024) by randomly masking tokens in the entire sequence using the uniformly sampled probability $p \sim U(0.5, 1)$. An alternative would be to sample a different masking probability per frame, similar to Chen et al. (2024), or leaving $k \sim U(0,T)$ frames unmasked to closely emulate inference.

	Parameter	Genie	Genie w/ prepend	Jafar	Jasmine-base
Tokenizer	# blocks	8			4
	# heads	8			
	model dim	512			
	ffn dim	512			2048
	# codes	1024			
	latent dim	32			
	patch size	16			
	total train steps	300k			
	learning rate	$3*10^{-4}$			
	lr decay end	$3*10^{-4}$			0
	batch size	48			
LAM	# blocks	8			4
	# heads	8			
	model dim	512			
	ffn dim	512			2048
	# codes	6			
	latent dim	32			
	patch size	16			
	total train steps	200k			
	learning rate	$3*10^{-5}$			
	lr decay end	$3*10^{-6}$			0
	batch size	48			
Dynamics	# blocks	12			6
	# heads	8			
	model dim	512			
	ffn dim	512			2048
	total train steps	200k			
	learning rate	$3*10^{-5}$			
	lr decay end	$3*10^{-6}$			0
	batch size	36			
	action conditioning	additive	prepend		prepend
	baseline	MaskGIT			
Training	optimizer	AdamW			
C	lr schedule	cos			wsd
	warmup steps	1k			
	wsd decay steps	_			10%
	dataset size (frames)	50M			
	co-training	yes		no	
Inference	temperature	1.0	·	<u>. </u>	<u> </u>
THETCHE	maskgit steps	25			

Table 6 | Configurations used in our experiments. We only show the difference to our base Genie configuration. Note that Bruce et al. (2024) uses a tokenizer patch size of four, and that we use an expanded dataset with 50M frames for all of our runs (to ensure that no method performs worse due to overfitting).

```
mask_prob = jax.random.uniform(rng1, minval=self.mask_limit)
mask = jax.random.bernoulli(rng2, mask_prob, vid_embed.shape[:-1])
mask = mask.at[:, 0].set(False)
vid_embed = jnp.where(jnp.expand_dims(mask, -1), self.mask_token, vid_embed)
Figure 8 | Code snippet from Willi et al. (2024) showing their batched masking logic.
mask_prob = jax.random.uniform(
    _rng_prob, shape=(batch_size,), minval=self.mask_limit
)
per_sample_shape = vid_embed_BTNM.shape[1:-1]
mask = jax.vmap(
    lambda rng, prob: jax.random.bernoulli(rng, prob, per_sample_shape),
    in_axes=(0, 0),
)(jnp.asarray(_rngs_mask), mask_prob)
mask = mask.at[:, 0].set(False)
vid embed BTNM = jnp.where(
    inp.expand dims(mask, -1), self.mask token.value, vid embed BTNM
)
```

Figure 9 | Code snippet from Jasmine showing its batched masking logic.

G. Evaluation Metrics

In early experiments, we found the Genie configuration to suffer from a discrepancy in performance between validation loss and autoregressive rollouts (Figure 2). Therefore, besides validation loss, Jasmine also tracks rollout metrics throughout training of the dynamics model. We generate a single frame using the sampling logic of the respective architecture and calculate SSIM and PSNR between the generated frame and the ground-truth. Although rollout metrics are calculated on a single frame, we find that they directly correlate with the model's performance in generating full rollouts. Validation and rollout metrics are calculated on a validation set. The rollouts in Figures 1, 4, 10 and 11 are sampled from frames of a test set.

H. Jafar's Batched Masking Logic

Whereas Willi et al. (2024) sample a single masking probability and apply the same masking pattern to all samples in a batch (Figure 8), Jasmine samples batch_size many sampling probabilities and uses per-sequence masking patterns (Figure 9). This leads to significantly reduced loss variance (Figure 3), especially in highly distributed settings.

I. Model Inspection using Treescope

Jasmine's training loop is highly modular and supports easy model surgery as well as model inspection using Treescope (Johnson, 2024). We provide a demo notebook⁵ alongside our repository, which illustrates debugging a common training instability (Figure 12).

⁵https://colab.research.google.com/drive/1zHkciFIZxXloJgue9F5LtFlAOm00rJIf

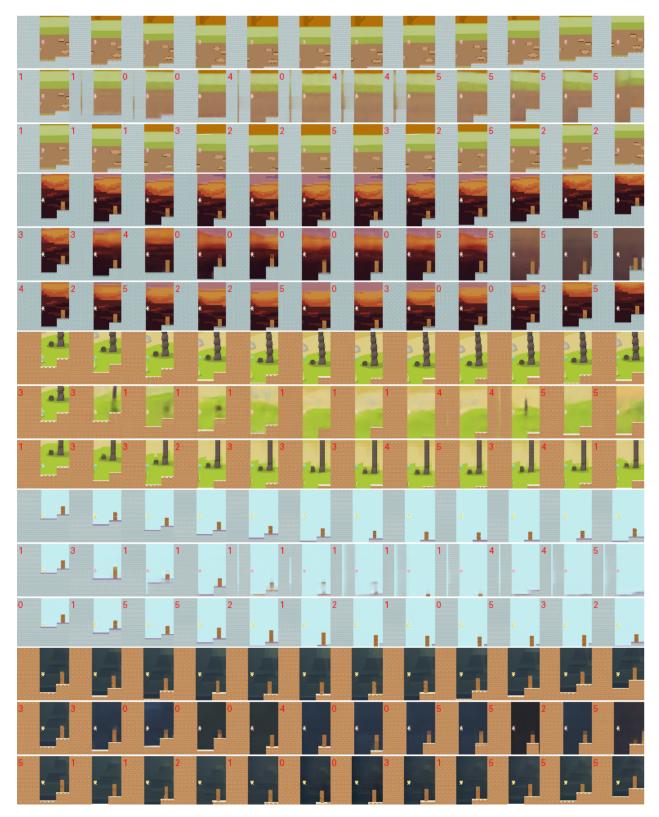


Figure 10 | Autoregressive rollouts on five randomly selected trajectories from the CoinRun environment. Each set of three rows corresponds to one trajectory, showing ground-truth frames (top), Jafar samples (middle), and Jasmine samples (bottom). The four conditioning frames are omitted.

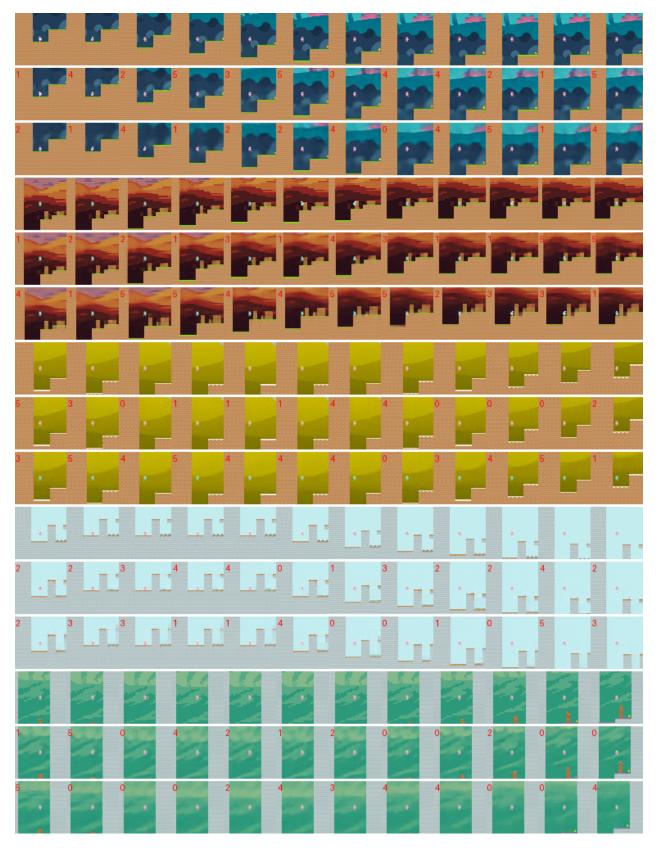


Figure 11 | Autoregressive rollouts on five randomly selected trajectories from the CoinRun environment. Each set of three rows corresponds to one trajectory, showing ground-truth frames (top), samples using the diffusion baseline (middle), and samples using the MaskGIT baseline (bottom). The four conditioning frames are omitted.

```
# checkpoint @ 20k
    lam, rng = build_model(args, rng)
    optimizer, lr_schedule_fn = build_optimizer(lam, args)
    step, optimizer, loader_iterator = restore_checkpoint_if_needed(
        args, ckpt_mgr, optimizer, loader_iterator, 20000
    lam = optimizer.model
    print(f"Restored optimizer and dataloader at step {step}.")
    enable_sowing(lam)
    print("Sowing enabled on encoder/decoder.")
    outputs = lam(batch, training=True)
    nnx.display(lam.encoder)
        num blocks=4,
<del>_</del>__
        dropout=0.0,
        param_dtype=⊳jax.numpy.float32, □
        dtype=⊳jax.numpy.bfloat16,
       sow_logits=True,
       sow_weights=False,
        sow activations=False,
        input_norm1=> LayerNorm(scale=Param(value=<jax.Array float32(768,) ≈1.0 ±0.0041 [≥0.99, ≤1.0] nonz
        input_dense= Linear(kernel=Param(value=<jax.Array float32(768, 512) ≈6.6e-05 ±0.036 [≥-0.091, ≤0.
        input norm2= LayerNorm(scale=Param(value=<jax.Array float32(512,) ≈1.0 ±0.0037 [≥0.98, ≤1.0] nonz
        pos_enc=> SpatioTemporalPositionalEncoding(d_model=512, max_len=5000, pe=Variable(value=<jax.Array
        blocks= [STBlock(activations=Intermediate(value=(<jax.Array float32(36, 16, 17, 512) ≈0.9 ±1.5 [≥
        output_dense=> Linear(kernel=Param(value=<jax.Array float32(512, 32) ≈0.00021 ±0.044 [≥-0.11, ≤0.1
        logits=vIntermediate( # 313,344 (626.7 KB)
          value=▼(
          v<jax.Array bfloat16(36, 16, 17, 32) ≈-0.013 ±6.2 [≥-4.4e+01, ≤4.2e+01] zero:1 nonzero:313_343</pre>
                  axis 1: 16
axis 2: 17
              axis 0: 36
                axis 3:
```

Figure 12 | Treescope visualization when performing model inspection. The notebook illustrates inspecting output logits at specific checkpoints.