Article

Not peer-reviewed version

ByteCraft: Generating Video Games and Animations Through Bytes

Alexia Jolicoeur-Martineau * and Emy Gervais

Posted Date: 26 March 2025

doi: 10.20944/preprints202503.1962.v1

Keywords: large language model; LLM; bytes; text-to-bytes; video-games

 doi:10.20944/preprints202503.1962.v1

Article

🎮ByteCraft: Generating Video Games and Animations Through Bytes

Alexia Jolicoeur-Martineau * and Emy Gervais

[1] Samsung SAIL Montréal
[2] Independent
* Correspondence: alexia.j@samsung.com

Abstract: Generating new video games using AI has potential to be the next holy grail of the video game industry. Current AI efforts have focused on two directions: i) controllable video generation and ii) code generated by Large Language Models (LLMs). The first direction is limited due to short-term memory and increasing corruption (blur, noise) over time. The second direction is promising, but it requires a lot of human hand-holding with human-provided assets. Generating hours of coherent interactive video content is infeasible. In this paper, we instead attempt the overly ambitious problem of end-to-end generation of Small Web Format (SWF) games and animations through bytes. By modeling bytes, one does not need code or assets to potentially obtain full games with title screen, narrative, text, graphics, music, and sounds. We make a first attempt by fine-tuning a 7-billion-parameter LLM at 32K context length to generate the bytes of video games and animations conditional on a text description. Our model (ByteCraft) can generate up to 32K tokens, each containing at most 4-5 bytes (generating files as big as 140 KB). Some of the generated files are partially working (4.8-12%), or fully working (0.4-1.2%). ByteCraft is a proof-of-concept highlighting what could be possible given more scaling and engineering effort. We open-source our model and inference code alongside a dataset of 10K synthetic prompts for use with ByteCraft. **Model**: https://huggingface.co/SamsungSAILMontreal/ByteCraft. **Inference Code**: https://github.com/SamsungSAILMontreal/ByteCraft.

Keywords: large language model; LLM; bytes; text-to-bytes; video-games

1. Introduction

Current methods for video game generation. While a lot of research is done on generating videos [1–3], very little effort is done on the video game front. Current state-of-the-art models for generating video games focus mainly on controllable video generation [4–9], which effectively consists of walking simulators with a few seconds of memory and increasingly blurry images over time. More recently, Large Language Models (LLMs) [10–14] have started being used to generate video games through code. This approach is promising, but generating complex games requires many rounds of human-AI interactions and user-provided graphics and audio assets.

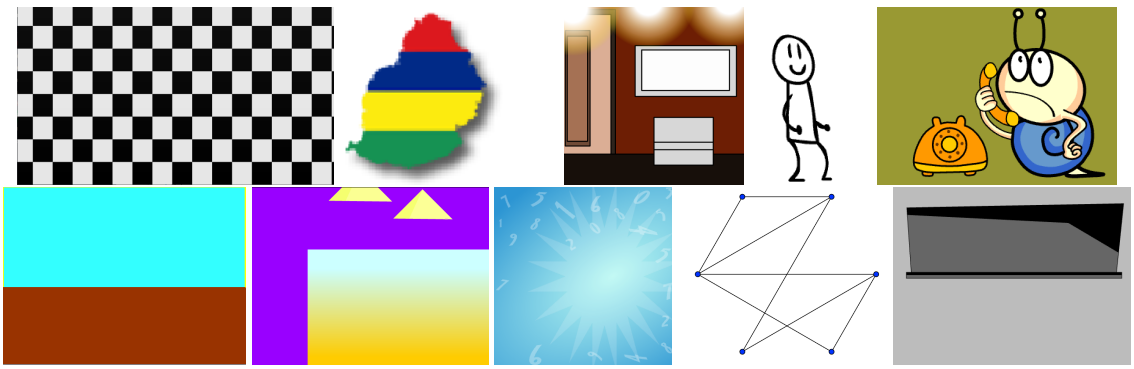

Figure 1. Screenshots of files generated by ByteCraft.

 doi:10.20944/preprints202503.1962.v1

A promising direction: directly generating bytes. A promising new line of direction is the generation of bytes from various types of files (e.g., images, videos, text, programs, etc.) using language models [15–17,17,18]. By staying in the byte world, one can generate any type of file found on a computer.

Why is byte generation not widespread? This direction has received very little attention for many reasons. First, simple files can take an enormous amount of bytes, leading to extremely large context lengths. For example, a simple 1Mb file requires 1 million byte tokens. Second, most modalities that people care about have specific neural network architecture and methods that have been developed and iterated over many years by researchers to make them excellent (e.g., images using 2D convolutions). Treating these modalities as bytes is challenging and may require many improvements in order to beat existing methods.

What are the problems associated with using bytes and how do we deal with them? The main difficulties when generating bytes of games and animations are 1) scaling (due to high-context length and limited data) and 2) overfitting. To handle longer context length than byte-level models, we tokenize bytes into 108K tokens, leading to approximately 2.29 bytes per token (with some extreme cases at 4-5 bytes per token). With a 32K sequence length, we can generate games of around 73 KB in size. To reduce the risks of overfitting, we produced multiple prompts per sequence of bytes.

The first-of-a-kind. In this work, we make the first attempt at building a generator of Small Web Format (SWF) [19] games and animations through bytes. SWF is a complex multi-modal format containing images, videos, code, fonts, and more data types. A single incorrectly placed byte could break the generated file. We are the first to tackle such a challenging task. We do so by fine-tuning an LLM (Qwen2.5-7B) to generate bytes conditional on a text prompt describing the game/animation, making our approach fully end-to-end. Our model, ByteCraft, was trained on limited resources (4 GPUs) for many months. It can generate up to 32K bytes (games/animations smaller than 73KB on average, with the largest we have seen at around 140KB).

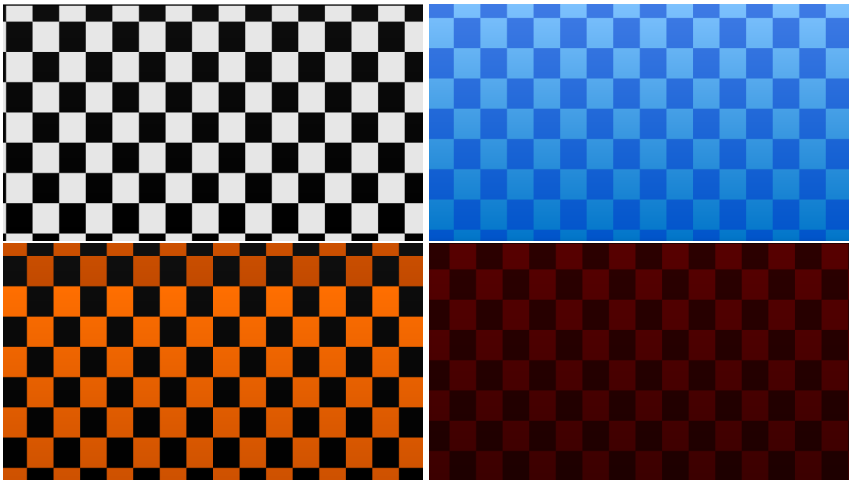

Figure 2. Checkered patterns in motion generated in different colors by ByteCraft.

2. ByteCraft

2.1. Architecture

We need to be able to condition on both text description and bytes. Since no model exists for generating bytes of video games and animations, this part has to be learned. LLMs are particularly good with unstructured text prompts.

We considered the following open-source LLMs as base model: Llama-3.1-8B [20], Ministral-8B [21], Qwen2-VL-7B [22], and Qwen2.5-7B [23]. In preliminary experiments, we found Qwen2.5-7B to be the best; thus, we used it as our base model.

 doi:10.20944/preprints202503.1962.v1

2.2. Tokenization

We used Byte Pair Encoding (BPE) [24] to encode the bytes of video games and animations into 108K new tokens containing on average 2.29 bytes per token (going up to 4-5 bytes per token in rare cases). This allowed us to scale to larger games than would be possible with byte-level generation. These new tokens were added to the tokens of the pre-trained model.

2.3. Data Augmentation on Prompts

Using Qwen2.5-7B, we generated multiple prompts per sequence of bytes in order to produce more diversity and reduce risks of overfitting.

2.4. Training

We fine-tuned ByteCraft with progressively bigger context length (4K, 8K, 32K) using AdamW [25,26]. For the early stages of training, we used the Muon optimizer [27], which accelerated training. Modern techniques such as Fully Sharded Data Parallel(FSDP) [28] and fused kernels [29] were used to accelerate training and reduce memory cost. Training took around 4 months in total using 4 GPUs. The model was trained until it reached a cross-entropy loss of 0.15 (Perplexity (PPL) of 1.16).

2.5. Usage

The user provides a text prompt to describe the video game or animation that they want to be generated. It can be written in any language. We provide some prompt examples in Figure 3. Once the prompt is given, ByteCraft generates k SWF files per prompt, where k is set by the user. These files can then be opened with the Ruffle player [30].

Using vLLM v0.7.3 [31], the model can simultaneously generate 25 files at 32K context length in around 10 minutes on a single A100 with 80Gb memory.

```
Please generate an animation with
the following specifications:
- Frame rate: 24.0
- Resolution: 500x400
- Language(s): English, Japanese
- Title: Bulletproof
- Size: 42KB
- Description: This is an animation
of stick figures shooting and
fighting in slow-motion with
martial art techniques.
```

```
Create a dress-up game, featuring
the character Emily, with a stylish
wardrobe. This is a single-player
game with a resolution of 800x600.
The game was released in 2007. Some
of the game features are changing
background, hair, eyes, mouth,
ears, and eyebrows, as well as
using stickers and patches to
decorate clothes and accessories.
```

Figure 3. Examples of prompts (Left: Structured, Right: Unstructured)

3. Results

We tested ByteCraft qualitatively and quantitatively after generating 250 files from 250 prompts from a held-out set.

For quantitative measures, we 1) calculate the maximum Jaro-Winkler string similarity [32] between the generated file and training files ("Max similarity"), 2) the Jaro-Winkler string similarity [32] between the generated file and the true file with the associated prompt ("Truth similarity"), and 3) the percentage of "fully broken" files as determined by verifying its header and metadata; if parsing fails, the file is likely invalid.

Since this is the first attempt at generating such files, assessing the quality of the files is tricky. We thus rely on qualitative measures by manually opening each file using the Ruffle player [30] (Nightly 2025-03-14). We categorize games that are not "fully broken" as i) "blank canvas" (with a flat background color), ii) "stuck on a loading screen", iii) "showing/hearing something" (this is subjective; it means that something is shown or heard and it is not a loading screen or blank canvas), iv) "Fully Working" (the file is working and playable from a quick assessment).

 doi:10.20944/preprints202503.1962.v1

The results are shown in Table 1. We see that most games are broken, but a few show something visually/audibly interesting, and a tiny percentage functions properly. On average, around 21% of the bytes in the files are novel, as determined by the maximum Jaro-Winkler similarity. The results show that the model can learn some patterns about bytes enough to produce some semi-working or working files.

Table 1. Results from 250 prompts.

Classification	$\min_p = 0.05$ $T = 0.1$	$\min_p = 0.05$ $T = 0.3$	$\min_p = 0.10$ $T = 0.3$	$\min_p = 0.20$ $T = 0.3$
Fully broken (wrong format)	20.4%	26.4%	20.4%	26.0%
Blank canvas (flat color)	69.6%	62.8%	67.2%	68.0%
Stuck on loading screen	4.4%	4.4%	4.0%	0.8%
Showing/hearing something	4.8%	5.6%	8.0%	4.0%
Fully working	0.4%	0.8%	0.4%	1.2%
Max similarity	0.790	0.790	0.789	0.791
Truth similarity	0.627	0.628	0.628	0.626

4. Potential Future Improvements

ByteCraft is a first attempt at generating open-web games and animations through bytes. However, it generates many broken files. In this section, we propose some improvements that could improve the performance of this method.

4.1. Scaling

ByteCraft was trained on extremely limited resources (4 GPUs over 4 months). Given large-scale resources, ByteCraft could improve in performance and be extended to generate larger games at higher context length.

4.2. Reinforcement Learning

To reduce the number of broken files, one could i) generate many files from various prompts with the current model, ii) have an automatic visual evaluator (possibly using a visual LLM such as the latest Qwen2.5-VL [33]) from a screenshot giving a reward between 0 and 1 for broken to fully working, and iii) use reinforcement learning to push the model toward generating valid working files or more aligned with the prompt.

4.3. Test-time Compute

Test-time scaling methods [34,35] have the potential to increase quality and diversity of the generated samples at inference without retraining.

4.4. Better Generalization on Small Data Using Data Augmentations on Bytes

Data augmentation is currently applied only to prompts: we generate a different prompt for each sequence of bytes. However, to truly enable the model to understand and generalize given a limited amount of training examples, one could i) randomly permute the order in which file components are stored (e.g., image 1, font1, code1, image2 $\rightarrow$ image2, image1, code1, font1). Other data augmentations are also possible, such as ii) having an AI rewrite the programming language code differently or iii) adding noise to the image assets contained in the file or changing their format (e.g., PNG [36] $\rightarrow$ JPEG [37]). The difficulty with all the data augmentation strategies proposed is that they require a good understanding of the file format, as one would need to determine where each asset starts and ends with their format and possibly where they are referred to in the code contained in the file. Ideally, this could be inferred directly from bytes, but it is possible that this would instead require expensive reverse engineering.

 doi:10.20944/preprints202503.1962.v1

5. Conclusions

We built ByteCraft, the first generator of games and animations through bytes. Contrary to existing approaches, ByteCraft is fully end-to-end. Given our limited resources (4 GPUs), it is limited to small files (32K context-length, around 73KB on average, but can be as big as 140KB). ByteCraft will require more scaling and could be extended to more types of games, including compressed versions of larger video games. This approach of generating files from text is not limited to video games and animations; it could be used to generate any kind of file end-to-end on a computer.

Parallel with early molecule generation. A parallel exists between ByteCraft and autoregressive molecule generation. Molecules can be represented as SMILES strings [38] and their context length is generally small (around 10-250 tokens without BPE). We show below some of the progress of molecule generation over time on the Zinc-250K dataset [39]:

1. GVAE [40]: 0.7% valid molecules (← ByteCraft is here)
2. CVAE [41]: 7.2% valid molecules
3. RVAE [42]: 34.9% valid molecules
4. GFVAE [43], STGG [44], and many others: 100% valid molecules, but not necessarily realistic and synthesizable
5. STGG+AL [45]: 100% valid molecules with high synthesizability and out-of-distribution properties (← a future ByteCraft version could be here)

In our case, we tackle the much harder problem of autoregressively generating SWF files with up to 32K tokens. Currently, some of the generated files are partially working (4.8-12%) or fully working (0.4-1.2%). We are thus at step 1, the equivalent of GVAE for molecule generation in 2016. Our end goal is generating 100% valid files with high novelty. We propose some potential directions of improvements in Section 4 on how to get there. We hope this crazy project inspires researchers and hobbyists toward the lofty goal of generating games through bytes.

References

1. Runaway. Runway Research | Gen-2: Generate novel videos with text, images or video clips, 2023.
2. OpenAI. Sora: Creating video from text, 2024.
3. DeepMind. Veo 2: A High-Definition Generative Model for Video, 2024.
4. Menapace, W.; Lathuiliere, S.; Tulyakov, S.; Siarohin, A.; Ricci, E. Playable video generation. In Proceedings of the Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 10061–10070.
5. Yang, M.; Li, J.; Fang, Z.; Chen, S.; Yu, Y.; Fu, Q.; Yang, W.; Ye, D. Playable Game Generation. *arXiv preprint arXiv:2412.00887* **2024**.
6. Valevski, D.; Leviathan, Y.; Arar, M.; Fruchter, S. Diffusion models are real-time game engines. *arXiv preprint arXiv:2408.14837* **2024**.
7. Che, H.; He, X.; Liu, Q.; Jin, C.; Chen, H. Gamegen-x: Interactive open-world game video generation. *arXiv preprint arXiv:2411.00769* **2024**.
8. Yu, J.; Qin, Y.; Wang, X.; Wan, P.; Zhang, D.; Liu, X. GameFactory: Creating New Games with Generative Interactive Videos. *arXiv preprint arXiv:2501.08325* **2025**.
9. Kanervisto, A.; Bignell, D.; Wen, L.Y.; Grayson, M.; Georgescu, R.; Valcarcel Macua, S.; Tan, S.Z.; Rashid, T.; Pearce, T.; Cao, Y.; et al. World and Human Action Models towards gameplay ideation. *Nature* **2025**, *638*, 656–663.
10. Todd, G.; Padula, A.G.; Stephenson, M.; Piette, É.; Soemers, D.; Togelius, J. GAVEL: Generating games via evolution and language models. *Advances in Neural Information Processing Systems* **2024**, *37*, 110723–110745.
11. Hu, C.; Zhao, Y.; Liu, J. Game generation via large language models. In Proceedings of the 2024 IEEE Conference on Games (CoG). IEEE, 2024, pp. 1–4.
12. Anjum, A.; Li, Y.; Law, N.; Charity, M.; Togelius, J. The ink splotch effect: A case study on chatgpt as a co-creative game designer. In Proceedings of the Proceedings of the 19th International Conference on the Foundations of Digital Games, 2024, pp. 1–15.
13. Rosebud AI. AI Game Creator | AI-Powered Game Dev Platform, 2024.
14. X. Grok 3 Beta — The Age of Reasoning Agents | xAI, 2025.

 doi:10.20944/preprints202503.1962.v1

15. Horton, M.; Mehta, S.; Farhadi, A.; Rastegari, M. Bytes are all you need: Transformers operating directly on file bytes. *arXiv preprint arXiv:2306.00238* **2023**.
16. Wu, S.; Tan, X.; Wang, Z.; Wang, R.; Li, X.; Sun, M. Beyond Language Models: Byte Models are Digital World Simulators. *arXiv preprint arXiv:2402.19155* **2024**.
17. Pérez, J.C.; Pardo, A.; Soldan, M.; Itani, H.; Leon-Alcazar, J.; Ghanem, B. Compressed-Language Models for Understanding Compressed File Formats: a JPEG Exploration. *arXiv preprint arXiv:2405.17146* **2024**.
18. Han, X.; Ghazvininejad, M.; Koh, P.W.; Tsvetkov, Y. Jpeg-lm: Llms as image generators with canonical codec representations. *arXiv preprint arXiv:2408.08459* **2024**.
19. Systems, A. *Adobe Flash Player Administration Guide for Flash Player 10.1*, 2010. Archived from the original (PDF) on 2010-11-21. Retrieved 2011-03-10.
20. Dubey, A.; Jauhri, A.; Pandey, A.; Kadian, A.; Al-Dahle, A.; Letman, A.; Mathur, A.; Schelten, A.; Yang, A.; Fan, A.; et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783* **2024**.
21. Mistral AI. Un Ministral, des Ministraux, 2024.
22. Wang, P.; Bai, S.; Tan, S.; Wang, S.; Fan, Z.; Bai, J.; Chen, K.; Liu, X.; Wang, J.; Ge, W.; et al. Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution. *arXiv preprint arXiv:2409.12191* **2024**.
23. Yang, A.; Yang, B.; Zhang, B.; Hui, B.; Zheng, B.; Yu, B.; Li, C.; Liu, D.; Huang, F.; Wei, H.; et al. Qwen2.5 Technical Report. *arXiv preprint arXiv:2412.15115* **2024**.
24. Gage, P. A new algorithm for data compression. *The C Users Journal* **1994**, *12*, 23–38.
25. Kingma, D.P. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* **2014**.
26. Loshchilov, I. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101* **2017**.
27. Jordan, K.; Jin, Y.; Boza, V.; You, J.; Cesista, F.; Newhouse, L.; Bernstein, J. Muon: An optimizer for hidden layers in neural networks, 2024.
28. Zhao, Y.; Gu, A.; Varma, R.; Luo, L.; Huang, C.C.; Xu, M.; Wright, L.; Shojanazeri, H.; Ott, M.; Shleifer, S.; et al. Pytorch fsdp: experiences on scaling fully sharded data parallel. *arXiv preprint arXiv:2304.11277* **2023**.
29. Hsu, P.L.; Dai, Y.; Kothapalli, V.; Song, Q.; Tang, S.; Zhu, S.; Shimizu, S.; Sahni, S.; Ning, H.; Chen, Y. Liger Kernel: Efficient Triton Kernels for LLM Training. *arXiv preprint arXiv:2410.10989* **2024**, [arXiv:cs.LG/2410.10989].
30. Ruffle. Ruffle - Flash Emulator, 2025.
31. Kwon, W.; Li, Z.; Zhuang, S.; Sheng, Y.; Zheng, L.; Yu, C.H.; Gonzalez, J.; Zhang, H.; Stoica, I. Efficient memory management for large language model serving with pagedattention. In Proceedings of the Proceedings of the 29th Symposium on Operating Systems Principles, 2023, pp. 611–626.
32. Jaro, M.A. Advances in record-linkage methodology as applied to matching the 1985 census of Tampa, Florida. *Journal of the American Statistical association* **1989**, *84*, 414–420.
33. Bai, S.; Chen, K.; Liu, X.; Wang, J.; Ge, W.; Song, S.; Dang, K.; Wang, P.; Wang, S.; Tang, J.; et al. Qwen2.5-vl technical report. *arXiv preprint arXiv:2502.13923* **2025**.
34. Snell, C.; Lee, J.; Xu, K.; Kumar, A. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314* **2024**.
35. Kim, H.; Choi, S.; Son, J.; Park, J.; Kwon, C. Neural Genetic Search in Discrete Spaces. *arXiv preprint arXiv:2502.10433* **2025**.
36. Roelofs, G. History of PNG. libpng, 2010. Retrieved 20 October 2010.
37. Collins English Dictionary. Definition of JPEG, 2013. Archived from the original on 21 September 2013. Retrieved 23 May 2013.
38. Weininger, D. SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules. *Journal of chemical information and computer sciences* **1988**, *28*, 31–36.
39. Sterling, T.; Irwin, J.J. ZINC 15–ligand discovery for everyone. *Journal of chemical information and modeling* **2015**, *55*, 2324–2337.
40. Gómez-Bombarelli, R.; Wei, J.N.; Duvenaud, D.; Hernández-Lobato, J.M.; Sánchez-Lengeling, B.; Sheberla, D.; Aguilera-Iparraguirre, J.; Hirzel, T.D.; Adams, R.P.; Aspuru-Guzik, A. Automatic chemical design using a data-driven continuous representation of molecules. *arXiv e-prints* **2016**, pp. arXiv–1610.
41. Kusner, M.J.; Paige, B.; Hernández-Lobato, J.M. Grammar variational autoencoder. In Proceedings of the International conference on machine learning. PMLR, 2017, pp. 1945–1954.
42. Ma, T.; Chen, J.; Xiao, C. Constrained generation of semantically valid graphs via regularizing variational autoencoders. *Advances in neural information processing systems* **2018**, *31*.

 doi:10.20944/preprints202503.1962.v1

43. Ma, C.; Zhang, X. GF-VAE: a flow-based variational autoencoder for molecule generation. In Proceedings of the Proceedings of the 30th ACM international conference on information & knowledge management, 2021, pp. 1181–1190.
44. Ahn, S.; Chen, B.; Wang, T.; Song, L. Spanning tree-based graph generation for molecules. In Proceedings of the International Conference on Learning Representations, 2021.
45. Jolicoeur-Martineau, A.; Zhang, Y.; Knyazev, B.; Baratin, A.; Liu, C.H. Generating π-Functional Molecules Using STGG+ with Active Learning, 2025, [arXiv:cs.LG/2502.14842].