An Inference Zoo for Real-Time Media Arts: *Avendish* as a Bridge Between AI and Creative Environments

Jean-Michaël Celerier
Société des Arts Technologiques
Montréal, Québec, Canada
jmcelerier@sat.qc.ca

Abstract

We present an extension of Avendish to the model inference domain. Our project uses an open-source C++ library to democratize real-time AI inference in real-time media arts environments by providing a unified interface to deploy contemporary machine learning models without the complexity of Python dependencies. Through an abstraction layer built on onnxruntime, Avendish enables artists to compile models into single, portable C++ libraries that integrate seamlessly with creative coding environments. The library currently supports 15 production-ready models spanning computer vision (BlazePose, DepthAnything2, YOLO variants), style transfer (StyleGAN, AnimeGAN family), emotion recognition, and language models (Qwen3, FastVLM). Model selection was informed by in-situ analysis at a major media arts research center, identifying the most requested AI capabilities among projects for two years. We demonstrate the library's effectiveness through its integration in ossia score and discuss how this approach addresses critical challenges in creative AI: reducing technical barriers, ensuring use in real-time contexts, and providing long-term preservability of artistic works that depend on AI models.

1 Introduction

The integration of artificial intelligence in media arts has reached an inflection point. Artists increasingly seek to incorporate sophisticated AI models into their creative practice, yet face significant technical barriers: managing Python environments, handling complex dependencies such as TensorRT, ensuring interactive real-time performance, and maintaining long-term stability of their works [Roads, 2023, McLean and Dean, 2023]. These challenges create a divide between the rapid advancement of AI research, led by technological platforms suitable for AI / ML researchers such as PyTorch and Tensorflow in Python, and its practical application in artistic contexts.

Creative coding environments such as Max/MSP, PureData, TouchDesigner, vvvv, and *ossia score* have long served as the primary tools for media artists to create interactive installations, live performances, and generative artworks [Puckette, 1991, Celerier et al., 2015]. While these environments excel at real-time audio-visual processing and interaction design, integrating contemporary AI models typically requires complex workarounds: external Python processes with asynchronous communication, network overhead, or platform-specific implementations that limit portability and increase latency. For instance, the StreamDiffusion inference system for real-time inference of Stable Diffusion and related models, is only implemented in these software through a separate Python wrapper launched as an external process and communicating through async pipes, which creates integration challenges especially for non-technical users. Another issue is the reliance on network AI inference, which is not always available: for instance, many exhibition spaces will not provide any kind of internet access to the computers running the artworks ; sometimes, the artworks may be presented in

Submitted to 39th Conference on Neural Information Processing Systems (NeurIPS 2025). Do not distribute.

remote locations without any possibility of internet at all, thus local inference is often the only viable option.

We leverage *Avendish*, a C++ library for abstraction of media processors (DSP algorithms, etc.) across host environments, to address these challenges and provide a unified interface for deploying AI models in a varied set of creative environments. Building on our previous work on multimedia plugin abstractions, *Avendish* extends the methodology to machine learning inference, enabling artists to compile AI models into single, portable C++ libraries by leveraging OnnxRuntime or other appropriate C++ ML libraries. This approach eliminates Python dependencies, ensures predictable real-time performance, and provides long-term preservability – critical for artistic works that must remain functional across sometimes decades by installing them on new computers.

Our contributions are threefold: First, we present a practical open-source framework for integrating onnxruntime-based inference into C++ creative coding environments with minimal overhead, leveraging compile-time reflection and modern C++ features to achieve zero-cost abstractions. Second, we provide a curated collection of 15+ AI models selected through analysis of real-world use cases at the Société des Arts Technologiques, ensuring that our selection addresses real artistic and creative studio needs rather than theoretical possibilities. Third, we discuss real-world adoption through integration in *ossia score* and deployment in artistic productions, validating our approach through actual use in professional creative contexts.

By lowering barriers to AI adoption in creative contexts, we enable new forms of artistic expression while addressing fundamental challenges in creative AI: How can we ensure that AI-dependent artworks remain functional over time? How can we provide artists with the real-time responsiveness required for live performance and how can we democratize access to AI capabilities without requiring extensive technical expertise?

The entire source code can be found in two repositories: `https://github.com/celtera/avendish` for AI model inference, and `https://github.com/ossia/score-addon-onnx` for the actual zoo.

2 Related Work

2.1 AI in Creative Coding Environments

The intersection of AI and creative coding has produced various approaches to integration. Wekinator [Fiebrink, 2009] pioneered accessible machine learning for artists through interactive machine learning, while ml.lib~[Bullock and Momeni, 2015] brought classical ML algorithms to Max/MSP and PureData. More recently, nn~ [Caillon and Esling, 2021] introduced more direct neural network support for Max/MSP and PureData, while TensorFlow.js enabled web-based creative ML applications [Smilkov et al., 2019].

These solutions either focus on classical specific ML algorithms (such as classification and regression for Wekinator), require complex-to-manage external dependencies (PyTorch for nn~), or sacrifice performance and ability to infer models on GPUs for accessibility. In addition, they do not operate at an abstraction suitable for inference of any kind of model in media arts environments. Some architectures require not just inference of a singular model file, but pre- and post- processing steps, or inference of more than a single model. Consider for instance large language models requiring tokenization steps, and encoding & decoding stages which may be all be provided through distinct models: the end-user only wants as affordance a single object with text and temperature inputs, and text output.

2.2 Model Deployment in Production

The challenge of deploying ML models in production environments has driven development of various frameworks. onnxruntime [ONNX Runtime developers, 2021] provides a cross-platform inference engine supporting multiple hardware accelerators APIs (Execution Providers). TensorRT [Vanholder, 2016] optimizes models for NVIDIA GPUs, while CoreML [Apple Inc, 2017] targets Apple platforms ; likewise, most platforms have their own custom inference API. For creative applications, these solutions typically require significant engineering effort to integrate, motivating our unified abstraction layer. Multiple Model Zoos exist in either proprietary ecosystems unsuitable to media artworks, or

Python environments: we can mention the Ailia SDK[1] and, in a way, the well-known ComfyUI environment.

2.3 Minimal-Dependency and Preservability

The concept of minimal-dependency software has gained traction in domains requiring long-term stability. In our previous work [Celerier, 2022], we demonstrated how compile-time reflection and modern C++ features enable plugin development without external dependencies: the software who wants to use an object implemented in Avendish does not require Avendish library code to be available at all: every type is just a standard C++ structure – no inheritance from a specific base type is for instance involved. A future software wishing to integrate a model developed with our system would just need to include the model's source files in their build system. This approach aligns with digital preservation principles [Rosenthal and Vargas, 2015], ensuring artistic works remain functional as technology evolves.

3 Design Principles

Our design philosophy for *Avendish* follows three core principles derived from extensive collaboration with media artists: First, **Minimal Dependencies**: Artists should not need to manage Python environments, version conflicts, or complex build systems. A model should compile to a single, self-contained library. Second, **Real-Time Performance**: Inference must be predictable and efficient enough for live performance contexts, with careful memory management and optional GPU acceleration. Third, **Preservability**: Artistic works using AI models should remain functional decades into the future, independent of external services or deprecated frameworks. Leveraging fixed C++ ecosystems with limited amount of dependencies enables easier preservation.

4 Architecture and Implementation

4.1 System Overview

Avendish provides a three-layer architecture for AI model integration. At the foundation, the Model Layer consists of ONNX models with metadata describing inputs, outputs, and preprocessing requirements, enabling a standardized representation of diverse AI models. These models are wrapped in a simple, C-like structure with specific fields for annotating the models' inputs and outputs. The middle Abstraction Layer employs C++ templates that generate type-safe bindings from the simple structure using compile-time reflection, ensuring that the overhead of our abstraction is eliminated during compilation. For instance, we are able to create a compile-time list of the input data types required by a given model implementation, which can then be used to generate the most efficient code for passing data from the host environment to the actual object. Finally, the Integration Layer provides host-specific adapters for creative coding environments, allowing seamless integration with Max/MSP, PureData, *ossia score*, and other platforms without requiring modifications to the core library.

4.2 Compile-Time Reflection for Zero-Cost Abstractions

A key innovation in *Avendish* is the use of modern C++ features to achieve zero-cost abstractions and extremely simple, readable object implementation. In particular, objects implemented in *Avendish* do not themselves have any dependency on an existing set of types or functions: the approach is purely declarative, based on the *shape* of the structure which is then reflected at compile-time and C++ concepts (in the PL meaning) which form a general ontology of useful media-arts related concepts.

As an example, coonsider this simplified version of inference of an emotion recognition model:

```
struct rgba_texture {
  enum format { RGBA };
  unsigned char* bytes;
  int width, height;
```

[1] https://axinc.jp/en/solutions/ailia_sdk.html

```
};
struct EmotionNet {
  static consteval auto name() { return "EmotionNet"; }
  struct inputs {
    struct { rgba_texture texture; } image;
    struct { float value; } min_confidence;
  } inputs;

  struct outputs {
    struct { std::optional<std::string> value; } main_emotion;
  } outputs;

  void prepare() { /* setup the ort session, allocate memory */ }

  void operator()(inputs& in, outputs& out) {
    /* onnxruntime inference setup for the given model */
    ort::Tensor in[1] = { tensor_from_rgba(inputs.image.texture); };
    ort::Tensor out[1];
    session.Run(in, out, ...);

    /* extract the data and output it from the node */
    if(out.emotions[0] > min_confidence) {
      outputs.main_emotion.value = "anger";
    }
  }
  Ort::Session session;
};
```

Through compile-time reflection, we automatically generate type-safe inputs and outputs with proper alignment, metadata for host environment integration, and zero-copy data paths where possible. This approach ensures that `sizeof(EmotionNet)` equals only the size of the ONNX session handle, inputs and outputs – no additional overhead for the abstraction. The compile-time nature of our approach contrasts sharply with traditional runtime-based plugin systems, which typically require virtual function calls, dynamic type checking, and heap allocations for parameter management.

4.3 Data types

The approach is able currently to handle as inputs and outputs types, any kind of standard C++ type, or type matching standard C++ interfaces: for instance, one can use as input of the objects types such as `float`, `int`, `double`, `char`, `std::string`, `std::vector`, `std::optional`, `std::variant` and any recursive combination of those, including in custom user-defined types: the output of our YOLO-Pose implementation is simply defined as:

```
struct Keypoint {
  int kp;
  float x, y;
};

struct DetectedYoloPose {
  std::string name;
  halp::rect2d<float> geometry; // rect2d is a simple struct { float x,y,w,h; };
  float probability{};
  std::vector<Keypoint> keypoints;
};
```

The binding back-end is able to turn this into the appropriate types for the host environment, at compile-time, by parsing the individual fields recursively and without additional annotation required from the object author. Unknown types that conform to the correct concepts would also work: one is able to use `boost::container::vector`, `boost::container::static_vector`, `boost::container::small_vector`, etc. The only requirement is that the type provides the basic `vector` operations: `size()`, `empty()`, `begin()`, `end()`, `push_back()`, and array access. For instance, in PureData and Max/MSP, output objects are converted into lists of `t_atom` types, the native data type of these environments.

Table 1: Current state of the *Avendish* Model Zoo

Category	Model	Use Case
Pose	BlazePoseBazarevsky et al. [2020] TRT-Pose YOLO-Pose Maji et al. [2022]	Single-person tracking Multi-person tracking Multi-person tracking
Style Transfer	AnimeGANv3 Liu et al. [2024]	Enhanced anime styling
Enhancement	FSR-GAN DeblurGANv2 Kupyn et al. [2019] DepthAnything2 Yang et al. [2024]	Super-resolution Motion deblurring Depth estimation
Generation	FBAnimeGAN[2] MobileStyleGAN Belousov [2021]	Anime-style generation Image generation
Detection	YOLO-blob Diwan et al. [2023] YOLO-segment Kang and Kim [2023]	Object detection Instance segmentation
Recognition	EmotionNet Gupta et al. [2021] ResNET He et al. [2020]	Facial emotion General classification
Language	Qwen3-8bYang et al. [2025] FastVLM Vasu et al. [2025]	Text generation Vision-language
Data processing	RapidLib Zbyszynski et al. [2017] regressor RapidLib classifier	Regression on simple datasets Classification on simple datasets

4.4 Model Collection

Our model selection process involved analysis of the projects done by our Innovation team at the Société des Arts Technologiques (SAT), a major media arts R&D center located in Montréal, QC. We identified recurring needs from multiple real-life creative projects we were tasked to provide support for, and selected models to add to our collection accordingly.

Table 1 shows the models currently supported. The selection prioritizes models frequently requested by artists, with emphasis on real-time capability, and where the base architecture has large applicability easily enabling custom models trained by the artists themselves (LLMs, ResNet, YOLO, StyleGAN, etc.). All the model arhcitectures are implemented through OnnxRuntime with the exception of the simple data regression and classification objects which enable fast Wekinator-like behaviour, well-suited for interactivity Hilton et al. [2021] directly within the host environment and which leverage the RapidLib C++ library.

An important focus was the ability to run inference on very low-power hardware: for instance on Raspberry Pi 5 without an additional AI hat ; the size of the models given to the inference architecture will of course be the primary driver for performance. We are able to run for instance BlazePose at consistently interactive FPS (above 20 FPS) on the Pi CPU.

5 Integration in Creative Environments

5.1 *ossia score* Integration

The primary deployment of *Avendish* is within *ossia score*, an intermedia sequencer designed for interactive installations and live performances. Models appear as nodes in the visual programming environment (Fig. 1) with automatic UI generation for parameters. AI inference can be synchronized with other media elements through the timeline system, and the sequencer handles model loading, memory management, and GPU resource allocation transparently.

5.2 Performance Considerations

Real-time performance requires careful attention to memory allocation and data flow. *Avendish* implements several optimizations to meet the requirements of live performance: all memory is

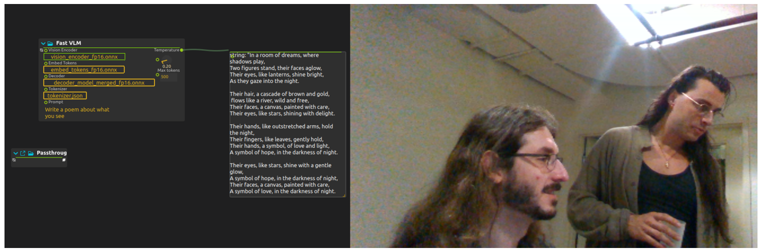

Figure 1: Fast-VLM integration in *ossia score*: the C++ Avendish node is compiled into an object where one can easily set input models, pass a real-time input video feed and get a poem as output entirely through an open-source visual dataflow system.

allocated at initialization to avoid runtime allocation overhead, lock-free queues ensure that the audio thread never blocks on inference operations, and asynchronous GPU transfers overlap with CPU processing to maximize throughput by leveraging worker threads whenever possible in actual object implementations. These design decisions stem from our experience with real-time multimedia systems, where even occasional frame drops or audio glitches can ruin an artistic performance.

6 Conclusion

While *Avendish* addresses multiple challenges in creative AI deployment, several limitations remain. The system is currently limited to ONNX format and specific additional C++ libraries, though this covers most production models and provides excellent cross-platform support. Nevertheless, for some model architectures, such as StyleGAN-related models, it has been comparatively hard to find suitable pre-trained models ; we had to translate models in their original PyTorch and/or TensorFlow format. Additionally, our focus on inference means that training within creative environments remains out of scope for now outside of a very simple case of regression training – though we argue that the separation of training and inference aligns with typical artistic workflows where models are trained offline and deployed for real-time use. Another caveat on our work is still the reliance on OnnxRuntime as core for the real-time inference mechanism, which is in itself a dependency. While much easier to manager than a complete Python virtual environment – only a few dynamic libraries need to be deployed to the customer, which may already be by the host environment: ossia score, TouchDesigner and Ableton Live already do.

In terms of future directions, our main focus is support for StreamDiffusion implementations to enable real-time live visuals, addressing a major request from VJs and live performers. One future addition to this would be support for specific tensor types, such as xtensor's `xt::tensor` types. This would open the door to a tensor-native graph environment where compatible tensors could be connected to each other at no conversion cost. Other models have been documented as important to have for media arts pipeline, but have yet to be implemented as Avendish nodes at the time of writing this paper: RAVE for audio inference, Whisper for speech-to-text and Kitten TTS for text-to-speech. Finally, currently, all Avendish back-ends are able to perform audio and data analysis, but image input and output is still to be developed for backends outside of ossia's which can be readily tested[3].

At large, *Avendish* demonstrates that the gap between AI research and creative practice can be bridged through careful system design. By prioritizing artist needs – ease of use, real-time performance, and long-term stability – we enable new forms of creative expression while maintaining the technical rigor required for production use. The compile-time reflection approach proves that zero-cost abstractions are achievable even for complex domains like ML inference. The growing adoption in the creative community validates our design decisions and points toward a future where AI tools are as accessible to artists as traditional media processing; this of course comes with the risks of misuse implicit to any democratization of a given technology: any artist could now implement pose-tracking surveillance devices on a Raspberry Pi from their favourite creative environment, instead of having to learn how to do it from e.g. Python.

[3] `https://ossia.io`

References

Apple Inc. Core ml framework, 2017. URL `https://developer.apple.com/documentation/coreml`.

V. Bazarevsky, I. Grishchenko, K. Raveendran, T. Zhu, F. Zhang, and M. Grundmann. Blazepose: On-device real-time body pose tracking. *arXiv preprint arXiv:2006.10204*, 2020.

S. Belousov. Mobilestylegan: A lightweight convolutional neural network for high-fidelity image synthesis. *arXiv preprint arXiv:2104.04767*, 2021.

J. Bullock and A. Momeni. Integrating machine learning with max/msp and pure data. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 265–270, 2015.

A. Caillon and P. Esling. Rave: A variational autoencoder for fast and high-quality neural audio synthesis. *arXiv preprint arXiv:2111.05011*, 2021.

J.-M. Celerier. Rage Against The Glue: Beyond Run-Time Media Frameworks with Modern C++. In *Proceedings of the International Computer Music Conference (ICMC)*, Limerick, Ireland, 2022.

J.-M. Celerier, P. Baltazar, C. Bossut, N. Vuaille, J.-M. Couturier, et al. OSSIA: Towards a unified interface for scoring time and interaction. In *Proceedings of the International Conference on Technologies for Music Notation and Representation (TENOR)*, 2015.

T. Diwan, G. Anirudh, and J. V. Tembhurne. Object detection using yolo: challenges, architectural successors, datasets and applications. *multimedia Tools and Applications*, 82(6):9243–9275, 2023.

R. Fiebrink. Wekinator: A system for real-time, interactive machine learning in music. In *Proceedings of The International Society for Music Information Retrieval*, volume 3, 2009.

V. Gupta, V. G. Panchal, V. Singh, D. Bansal, and P. Garg. Emotionnet: Resnext inspired cnn architecture for emotion analysis on raspberry pi. In *2021 International Conference on Recent Trends on Electronics, Information, Communication & Technology (RTEICT)*, pages 262–267. IEEE, 2021.

F. He, T. Liu, and D. Tao. Why resnet works? residuals generalize. *IEEE transactions on neural networks and learning systems*, 31(12):5349–5362, 2020.

C. Hilton, N. Plant, C. Gonzalez Diaz, P. Perry, R. Gibson, B. Martelli, M. Zbyszynski, R. Fiebrink, and M. Gillies. Interactml: Making machine learning accessible for creative practitioners working with movement interaction in immersive media. In *Proceedings of the 27th ACM Symposium on Virtual Reality Software and Technology*, pages 1–10, 2021.

C. H. Kang and S. Y. Kim. Real-time object detection and segmentation technology: an analysis of the yolo algorithm. *JMST Advances*, 5(2):69–76, 2023.

O. Kupyn, T. Martyniuk, J. Wu, and Z. Wang. Deblurgan-v2: Deblurring (orders-of-magnitude) faster and better. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 8878–8887, 2019.

G. Liu, X. Chen, and Z. Gao. A novel double-tail generative adversarial network for fast photo animation. *IEICE TRANSACTIONS on Information and Systems*, 107(1):72–82, 2024.

D. Maji, S. Nagori, M. Mathew, and D. Poddar. Yolo-pose: Enhancing yolo for multi person pose estimation using object keypoint similarity loss. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2637–2646, 2022.

A. McLean and R. T. Dean. *The Oxford handbook of algorithmic music*. Oxford University Press, 2023.

ONNX Runtime developers. Onnx runtime, 2021. URL `https://onnxruntime.ai/`.

M. Puckette. Combining event and signal processing in the max graphical programming environment. *Computer Music Journal*, 15(3):68–77, 1991. doi: 10.2307/3680767.

C. Roads. *Living Electronic Music*. MIT Press, 2023.

D. S. Rosenthal and D. L. Vargas. Emulation & virtualization as preservation strategies. In *The Andrew W. Mellon Foundation*, 2015.

D. Smilkov, N. Thorat, Y. Assogba, A. Yuan, N. Kreeger, P. Yu, et al. Tensorflow.js: Machine learning for the web and beyond. In *Proceedings of Machine Learning and Systems*, volume 1, pages 309–321, 2019.

H. Vanholder. Efficient inference with tensorrt. In *GPU Technology Conference*, volume 1, pages 1–24, 2016.

P. K. A. Vasu, F. Faghri, C.-L. Li, C. Koc, N. True, A. Antony, G. Santhanam, J. Gabriel, P. Grasch, O. Tuzel, and H. Pouransari. Fastvlm: Efficient vision encoding for vision language models, 2025. URL `https://arxiv.org/abs/2412.13303`.

A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

L. Yang, B. Kang, Z. Huang, Z. Zhao, X. Xu, J. Feng, and H. Zhao. Depth anything v2. *Advances in Neural Information Processing Systems*, 37:21875–21911, 2024.

M. Zbyszynski, M. Grierson, M. Yee-King, and L. Fedden. Write once run anywhere revisited: machine learning and audio tools in the browser with c++ and emscripten. 2017.

NeurIPS Paper Checklist

[Yes] , [No] , or [NA] .

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes] .

   Justification: The main claims in the abstract and introduction are that the paper presents a C++ framework, Avendish, for real-time, dependency-minimal AI inference in creative software, supported by a curated set of model architectures. These claims are consistently supported throughout the paper: Section 4 details the technical architecture, including the use of compile-time reflection and ONNX. Section 4.4 and Table 1 present the promised model architecture zoo and its selection rationale. Section 5 validates the real-world application through its integration with ossia score. The paper's scope is accurately represented, and its contributions are clearly substantiated in the corresponding sections.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The first three paragraphs of Section 6 discusses the limitations, both on a conceptual level (issues caused by choosing ONNX as a main platform) and practical level (not all useful models for media arts are implemented yet, the model zoo is an ongoing work).

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.

- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [NA]

   Justification: The paper focuses on practical approaches to model inference and software engineering rather than on problems of a more theoretical nature such as theorems.

   Guidelines:

   - The answer NA means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The entire source code is open, accessible and documented, as well as the build procedure for the ossia software, which will build the current model zoo. Build scripts and documentation are available for Windows, Linux and macOS. The only requirement is a very recent C++23 compiler toolchain (for instance Xcode 16.4 on macOS or clang-20 on Linux and Windows).

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.

(b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.

(c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: Links to the repositories are provided in introduction.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so No is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: [NA]

Justification: No model was trained as part of this research.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [NA]

Justification: The paper does not have experiments of a statistical nature.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Requirements for running model inference with our framework are discussed in section 4.3 and 4.4.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [NA]

   Justification: The research did not involve human subjects.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [Yes]

Justification: The introduction and conclusion discusses the impact for artists which is mainly about democratizing and making easier to use inference systems for models relevant to media arts practices.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

Answer: [NA]

Justification: The work does not introduce a new model. All the models mentioned are already open-weights, freely and easily downloadable from standard repositories such as HuggingFace and Github. It simply makes them more accessible to non-technically-versed users.

Guidelines:

- The answer NA means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: The author wrote all the avendish code and the model integration code. No models are provided directly by our zoo: the end-user has to download and pass themselves the model files to the objects developed as part of this work.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: The avendish library has a clear and explicit documentation linked on its Github page.

Guidelines:

- The answer NA means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [NA]

Justification: The paper did not crowdsource data from human subjects. The main source of data is the list of projects our organization has been providing technical support for, the list of requests for consulting we got and the technologies which were used in each project (e.g. N projects involving Whisper, M projects involving StreamDiffusion, etc.).

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: Same justification as above ; in addition, as a small, private non-profit organization we do not have yet an institutional review board.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [NA]

    Justification: LLMs were not used in any particular way for this research.

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.