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

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Creative AI Track.

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]	Single-person tracking
	YOLO-Pose Maji et al. [2022], RTM-Pose, ViT-Pose	Multi-person tracking
Style Transfer	AnimeGANv3 Liu et al. [2024]	Enhanced anime styling
Enhancement	FSR-GAN	Super-resolution
	DeblurGANv2 Kupyn et al. [2019]	Motion deblurring
	DepthAnything2 Yang et al. [2024]	Depth estimation
Generation	FBAnimeGAN[3]	Anime-style generation
	MobileStyleGAN Belousov [2021]	Image generation
Detection	YOLO-blob Diwan et al. [2023]	Object detection
	YOLO-segment Kang and Kim [2023]	Instance segmentation
Recognition	EmotionNet Gupta et al. [2021]	Facial emotion
	ResNET He et al. [2020]	General classification
Language	Qwen3-8bYang et al. [2025]	Text generation
	FastVLM Vasu et al. [2025]	Vision-language
Diffusion	StreamDiffusion, Flux, img2img-turbo	img2img, txt2img
Data processing	RapidLib Zbyszynski et al. [2017] regressor	Regression on simple datasets
	RapidLib classifier	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[2] 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.

[2]Implementation is fully open on Github: ossia/score-addon-onnx and ossia/score-addon-libre diffusion

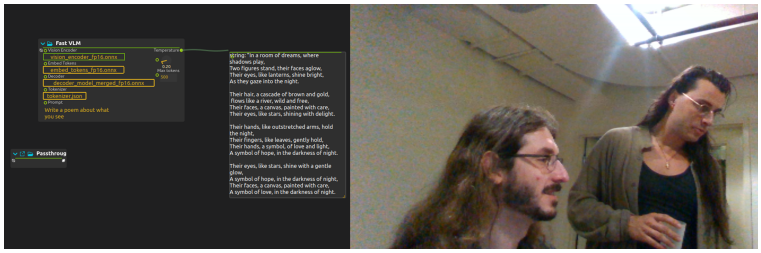

Figure 1: Fast-VLM integration in *ossia score*: the C++ Avendish node is compiled into an object where one can easily set input models, pass a real-time input video feed and get a poem as output entirely through an open-source visual dataflow system.

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 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[4].

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

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

devices on a Raspberry Pi from their favourite creative environment, instead of having to learn how to do it from e.g. Python.

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.