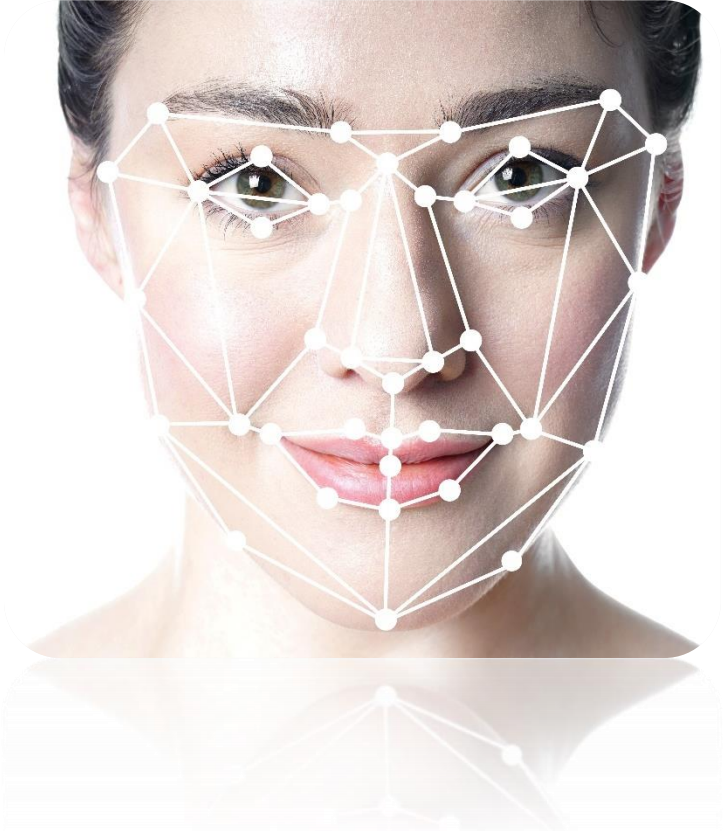


From Fragmentation to Unification: SOTA GO for Visual Graph Orchestration

Whitepaper 1.0

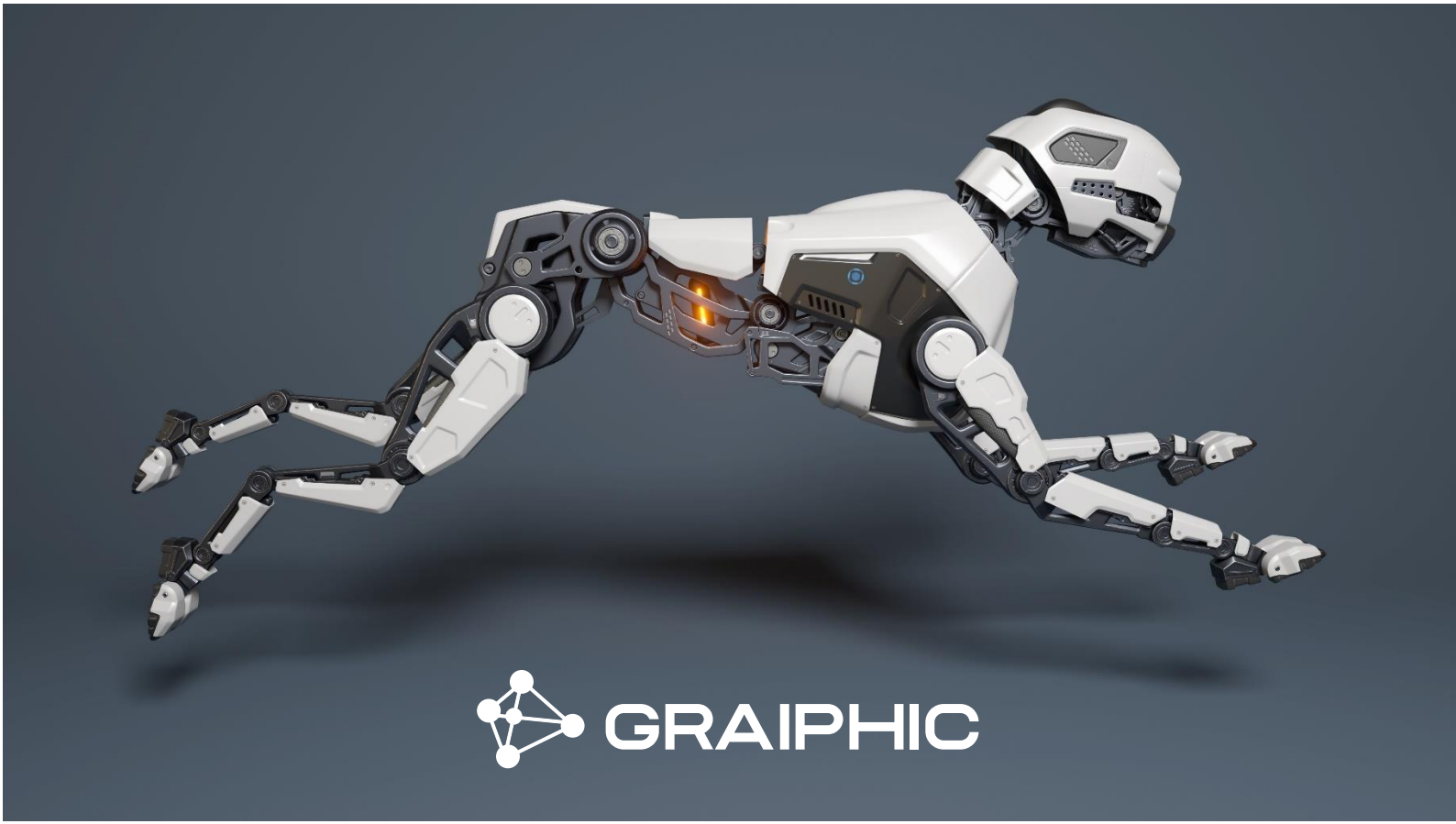


Table of Contents

Versioning

This document is subject to version control to ensure full traceability of changes. Each update is recorded with its author, date, and a short description of the modifications.

Version	Date	Author	Organization	Change Description
1.0	2025/12/13	Youssef Menjour	Graiphic	First publication of the SOTA GO Whitepaper

Preface — How This Whitepaper Fits into the Graiphic Program

This whitepaper is part of a coordinated series of four technical documents published by Graiphic to present the full scope, structure, and ambition of its unified AI and graph orchestration program.
Rather than isolated publications, these documents are designed to be read as a coherent whole. Each whitepaper focuses on a specific layer of the same technological stack, while sharing a common foundation built on ONNX, ONNX Runtime, and graphical graph orchestration inside the SOTA ecosystem.

The four whitepapers are:
SOTA — The LabVIEW IDE for Graph Computing: This foundational whitepaper introduces **SOTA**, the first building block of Graiphic's technology stack and a fully functional platform already available to industry, research laboratories, and academic institutions. It is the unified graphical environment developed by Graiphic to author, edit, train, optimize, and deploy ONNX graphs visually inside LabVIEW, and it defines the common execution model, tooling, and orchestration principles used across all other initiatives.

GO HW — From Models to Systems: This whitepaper focuses on extending ONNX-based graphs beyond AI inference into full system and hardware orchestration. It introduces hardware primitives (GPIO, DMA, ADC/DAC, timers, energy monitoring) as first-class graph nodes, enabling deterministic execution of AI-driven systems directly on CPUs, GPUs, FPGAs, NPUs, and embedded SoCs.

GO GenAI — From Fragmentation to Orchestration: This whitepaper addresses the fragmentation of today's Generative AI ecosystem. It transforms ONNX into a dynamic orchestration fabric capable of coordinating models, tokenizers, logic, data streams, and heterogeneous hardware runtimes inside a single executable graph, fully integrated within the SOTA environment and without reliance on Python glue code.
GO IML — From Theory to Superiority: This whitepaper introduces **Informed Machine Learning** as a native graph-based capability. It enables data-driven learning to be augmented with physics, logic, constraints, and expert knowledge directly inside ONNX training graphs, delivering faster convergence, stronger generalization, explainability, and deployability across real-world hardware.
Together, these four documents describe a single, unified vision: **one graph as the source of truth, one runtime for execution, and one graphical cockpit to design, orchestrate, and deploy intelligent systems from models to hardware.**
The complete Graiphic GO Whitepaper Series is available here:
https://github.com/Graiphic/GO-Whitepapers

Executive Summary

Artificial intelligence is evolving rapidly, yet its development remains fragmented and difficult to use at scale. Engineers must navigate a maze of tools. Researchers struggle to reproduce results. Companies lose time adapting models across frameworks and hardware. Although AI is everywhere, its creation process is still complex, scattered, and costly.

SOTA was built to transform this landscape. It offers a unified suite of tools, each designed with a strong focus on innovation and engineered for developers, researchers, and industrial teams. In essence, SOTA is to artificial intelligence and robotics what Microsoft Office is to productivity software, a coherent integrated environment that streamlines every stage of development.

At Graiphic, we imagined a world where AI development becomes unified, intuitive, and enjoyable. A world where a single graphical language allows anyone to design, train, optimize, and deploy intelligent systems. A world where ONNX is no longer a static file format but a dynamic foundation for orchestration.

SOTA turns that vision into a practical and powerful ecosystem.

It brings together the simplicity of graphical programming, the performance of ONNX Runtime, and the industrial reliability of LabVIEW. With SOTA suite, tasks that once required multiple tools and specialized teams can be completed inside one coherent environment. Prototyping becomes faster, deployment becomes easier, and innovation becomes accessible to everyone.

SOTA suite includes deep learning, computer vision, acceleration, and generative AI toolkits, along with tools such as FIG, Builder, and the Annotation Tool. Users can import models, prepare datasets, train neural networks, execute them on any hardware, and integrate them into real industrial architectures with unprecedented clarity. Everything works together, everything is optimized, and everything is expressed visually.

The advantages are immediate. Prototyping time decreases by up to half. Development costs are reduced significantly. Deployment becomes seamless across GPU servers, ARM devices, and FPGA technologies. SOTA replaces fragmentation with a unified experience that is both practical and inspiring.

More importantly, SOTA is the technological foundation that prepares the arrival of GO HW, GO GenAI, and GO IML. These future technologies will elevate ONNX into a dynamic graph capable of orchestrating logic, intelligence, and hardware inside a single, sovereign system.

SOTA is not only an innovative AI platform. It is a new way of thinking about how intelligent systems are created. It brings clarity where complexity existed, harmony where fragmentation dominated, and creativity where constraints once limited progress.

SOTA is the beginning of a unified AI world.

Introduction

Artificial intelligence has reached an extraordinary level of maturity, yet the journey from idea to real-world application remains surprisingly difficult. Behind every model lies a complicated sequence of tools, frameworks, scripts, conversions, and deployment challenges. Engineers spend more time orchestrating their environment than developing solutions. Researchers lose hours switching between frameworks. Companies struggle to bring prototypes into production because nothing speaks the same language.

- This fragmentation slows innovation everywhere.
- The problem is not the intelligence of the models.
- The problem is the complexity of the path that leads to them.

SOTA suite was created to remove this barrier.

Our goal was simple to state and difficult to achieve. We wanted to offer one environment that covers the entire lifecycle of AI. A place where models flow naturally from design to deployment. A place where engineers and researchers can work together with the same graphical language. A place where ONNX models are not just files but living components that interact, evolve, and adapt across hardware and software.

The result is a complete ecosystem built on three ideas.

First, **accessibility**. AI should not require mastering a dozen incompatible tools. SOTA provides a unified and intuitive graphical experience that makes complex architectures easy to understand and easy to build.
Second, **interoperability**. The choice of ONNX Runtime ensures that every model can move freely across frameworks and hardware without rewriting code. This guarantees long-term stability, transparency, and industrial reliability.
Third, **acceleration**. SOTA transforms the development process by reducing prototyping time, simplifying deployment, and enabling high-performance execution on GPU servers, industrial controllers, and embedded devices.
SOTA is not a new framework added to the existing landscape. It is the foundation that the landscape was missing. It brings coherence, clarity, and efficiency to a world that has expanded too fast and without structure.

This introduction opens the door to the story of how SOTA was built, why it responds to an urgent need, and how it prepares the arrival of GO HW, GO GenAI, and GO IML. Together, they form a new technological frontier where intelligence, logic, and hardware finally converge.

Graphs made complex systems visible. The next step is to let the same graph do more than predict. From inference to training and orchestration, it is still a graph.

From a static file to a living graph

Every ONNX model is a little play. The cast are nodes, the props are tensors, and the script is the graph. Most of us only hire the math stars that do convolutions, matmuls, and activations. Three quiet actors wait in the wings: If, Loop, and Scan. They rarely get called when we only do classic deep learning inference, yet they hold the keys to choreography. With them, a graph can describe not just what to compute, but when to compute and how often to repeat. That is where the story gets interesting.

We call this idea GO, for **g**raph **o**rchestration. GO is the goal of turning ONNX from a static description into a dynamic technology. The artifact stays the same, yet the way we use it changes. ONNX brings a universal, interoperable format. ONNX Runtime brings an efficient execution engine. Together they already run fast on many targets. With GO, the graph also carries schedules and control flow in a first-class way, so you coordinate learning loops, evaluation passes, and model lifecycle without leaving the ONNX world.

There is a catch. ONNX today shines as a file format and as a runtime target, yet it lacks a native editor. You can convert from popular frameworks, you can execute with ONNX Runtime, but you cannot comfortably import, edit, and create ONNX graphs without going back to third party toolchains. That dependency keeps ONNX as an excellent tool, not yet a complete graph computing framework. It slows innovation because the ideas must pass through a different language before they become ONNX.

This is the gap Graiphic is closing. We keep ONNX as the single source of truth, expose both levels of abstraction, and make the control flow nodes easy to use. Engineers can work at a Keras style layer level. Researchers can sculpt at the node level. Everyone edits the same graph, saves the same format, and benefits from the same runtime.

ONNX in plain language

Before we orchestrate anything, let us make the building blocks feel familiar. ONNX is an open way to write a computation as a graph. A node is an operation. An edge carries a tensor from one node to the next. Some tensors are not inputs at all but weights stored inside the model. Each node has attributes that set its behavior, for example a kernel size or an activation choice. Put these pieces together and you have a recipe the computer can follow step by step.

Think of ONNX as sheet music. The notes are operations like MatMul, Conv, Add, Relu. The bars are tensors that flow across the page. The tempo is set by shapes and types that tell the runtime how large the arrays are and how they line up in memory. A model file simply packages the score with its instruments. It contains the graph, the weights, the operator set version, and a little metadata such as names and documentation. You can pass this file between tools and keep meaning intact.

Why is this useful beyond conversion? Because a graph is precise and inspectable. You can open it, count the tensors, check the shapes, and see exactly how data moves. You can split it in two, reuse a prefix, or swap a small part without touching the rest. You can

run it on a laptop, a workstation, or a small board and expect the same logical behavior. When a team says one source of truth, this is what they mean.

If, Loop, and Scan enable control flow inside the graph, essential for orchestration and training logic. They're already part of ONNX and will be key to express full execution schedules. If choose a path based on a condition. Loop repeats a subgraph and carries state across steps. Scan walks over a sequence and collects results. Most people ignore them when they only deploy a fixed network, yet they make the format future ready. They are the handles we will use later to express schedules and learning cycles inside the same artifact.

A final detail completes the picture. ONNX is neutral about taste. It does not force a style like layers only or operators only. You can treat the graph as a high-level model if that is the right abstraction for an engineer, or you can treat it as a set of fine-grained operators if you are a researcher crafting something bespoke. The file does not change, only the editor you prefer.

Cheat sheet

- *Node: a single operation that consumes and produces tensors*
- *Tensor: an array with a shape and a data type*
- *Initializer: a tensor stored in the model, usually a weight*
- *Attribute: a small setting attached to a node*
- *Opset: the versioned catalog of available operators*

ONNX Runtime in practice

Now that the score is clear, meet the conductor. ONNX Runtime reads the graph, plans the work, and plays it efficiently on real hardware. It chooses kernels, arranges memory so tensors land where they should, and removes extra steps by fusing compatible nodes. The result is a compiled session that you can call many times with stable latency and a predictable footprint.

Think of execution as a three-part routine. First comes analysis. The runtime checks shapes and data types, folds constants, and prunes dead branches. Second comes partitioning. Subgraphs are assigned to Execution Providers that know how to run them fast, for example above the native CPU, CUDA and TensorRT for NVIDIA, oneDNN and OpenVINO for Intel, ROCm and VitisAI for AMD, and DirectML for Windows GPUs. Third comes scheduling. The runtime builds an execution plan that minimizes copies, aligns layouts, and reuses memory arenas so nothing is allocated in the hot path.

A useful detail is that the graph stays the source of truth. You can inspect the optimized graph, see which parts got fused, and verify exactly which provider runs which segment. If a device is missing, the same artifact still runs on a plain CPU provider with the same logical behavior, just at a different speed, that's what we call the Fallback mechanism. This keeps experiments honest and production portable.

Here is a simple way to place ONNX Runtime in your mental map.

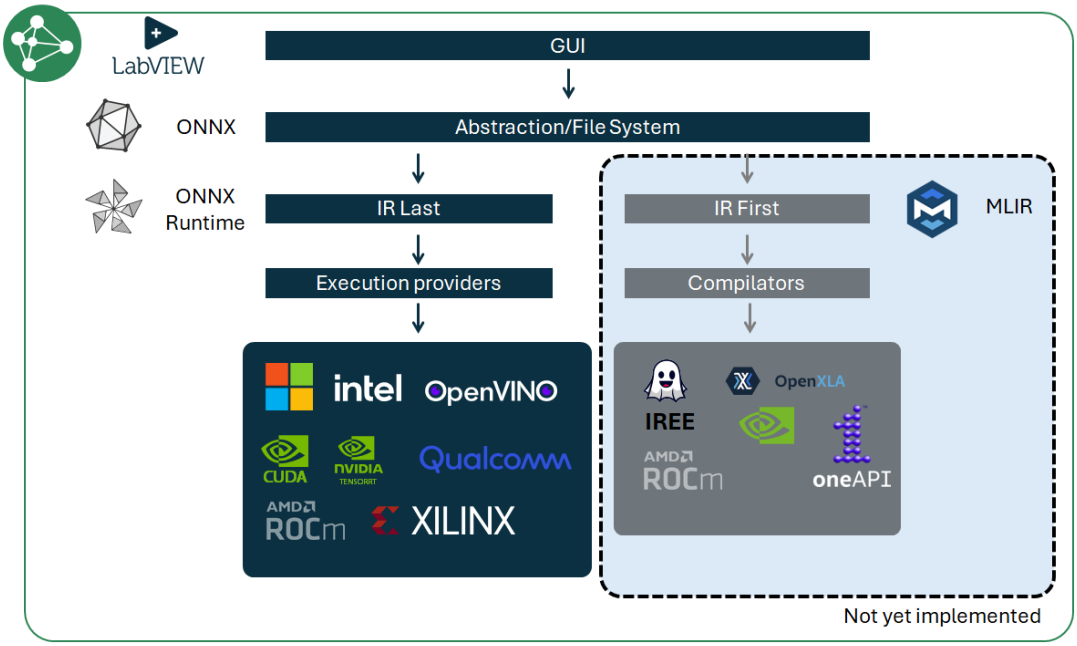
Library vs runtime, in one glance

- *A library gives you individual operations such as MatMul or Conv (examples: CUDA, OneDNN, RocM, DirectML)*
- *A runtime takes a whole graph, compiles it end to end, and decides how and where to run each part (examples: TensorRT, OpenVINO Runtime, VitisAI Runtime, ONNX Runtime)*
- *Libraries are the instruments, the runtime is the conductor*
- *ONNX Runtime integrates with many runtimes and libraries via Execution Providers and falls back to the CPU provider when needed*

Two side effects matter in practice. Efficiency improves because the runtime can fuse chains of operations and keep data in the right format between them. Energy improves because fewer copies and fewer cache misses translate into less wasted work. Both effects show up the moment you repeat inference at scale.

Compiler strategy: IR-last today, IR-first optional tomorrow.

An *Intermediate Representation (IR)* is the neutral "sheet music" of a program that enables analysis, optimization, and lowering to hardware. Today we favor an **IR-last** path with **ONNX Runtime**: the ONNX graph remains the source of truth while the runtime performs graph-level optimizations and **partitions subgraphs to hardware Execution Providers** (TensorRT, OpenVINO, DirectML, ROCm, ...). This maximizes portability, coverage, and time-to-first-inference. In parallel, we open an **IR-first** lane with **MLIR**: models are lowered through dialects (e.g., ONNX or StableHLO) and compiled end-to-end (e.g., via **IREE** or **OpenXLA**) for ahead-of-time specialization, tight latency budgets, and target-specific scheduling. **Bridges** (ONNX-MLIR, StableHLO) make both lanes interoperable. Net effect: under the same LabVIEW GUI and orchestration, users can pick **runtime breadth** (IR-last) or **compiler-grade specialization** (IR-first) per deployment.



Dual IR Strategy — ORT (IR-last) vs MLIR (IR-first) under LabVIEW Orchestration

Graiphic deliberately starts with an **IR-last** path built on **ONNX Runtime**: the ONNX file remains the single source of truth, while the runtime performs graph-level optimizations and **partitions subgraphs to hardware Execution Providers** before scheduling them efficiently. This maximizes portability, coverage, and time-to-first-inference. Next, we will expose an **optional IR-first lane** based on **MLIR** (e.g., IREE/OpenXLA). When targets or workloads benefit from ahead-of-time specialization, static-shape lowering, or custom dialects, the same ONNX graph can be lowered to MLIR dialects and compiled end-to-end to native executables. **Users will choose per deployment** between the ORT path (EP-driven runtime) and the MLIR path (compiler pipeline), under the same LabVIEW cockpit, device profiles, and monitoring plane. This dual strategy keeps our **portability-first** promise while opening the door to **compiler-grade determinism and specialization** where it matters.

Graph computing as a quiet revolution

Once you see a model as a graph, you start seeing most workflows as graphs. A graph is a contract that lists the steps, the data that flows between them, and the rules that govern the journey. It is transparent, easy to inspect, and easy to test. You can run a single subgraph to debug an issue, then run the full plan with the confidence that the behavior will match. This mindset turns scattered scripts into a single artifact that you can reason about.

With If, Loop, and Scan, graphs can carry full training, evaluation, and control loops — all in a reproducible, inspectable way.

Graphs also make optimization a first-class activity. Because the plan is explicit, a runtime can fuse operations, reuse buffers, and select precisions that fit the budget. Because the plan is versioned, a team can review changes, compare metrics, and roll back without guessing which script or notebook drifted. Provenance stops being a headache and turns into a property of the file.

This is why we describe GO as **g**raph **o**rchestration. The idea is simple. Keep one artifact. Put both computation and schedule inside it. Let the runtime turn that plan into an execution that is fast and predictable on real machines. You gain portability, you gain performance, and you gain a common language between engineers and researchers.

Micro checklist for graph ready work

- *Preprocessing and metrics belong in the graph when possible*
- *Control flow is explicit, not hidden in outer scripts*
- *Seeds, shapes, and dtypes are recorded for reproducibility*
- *Subgraphs are modular so teams can reuse and swap them*
- *The optimized graph is inspected like code*

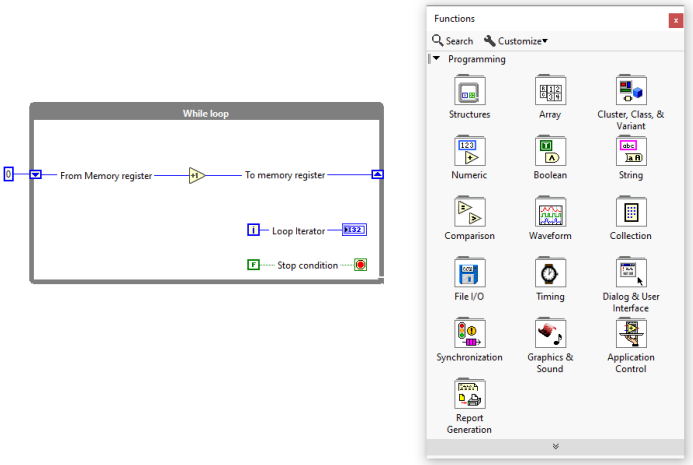
If graphs make complex systems visible, LabVIEW proved it decades ago by turning dataflow into an everyday tool for engineers. Let us rewind to see why that matters now.

Back to the 80s, NI and the birth of LabVIEW

Picture the mid-Eighties. Personal computers get a proper graphical interface. Engineers want instruments to talk to software without wrestling with arcane drivers. National Instruments sells the boards and sees the gap.

In Austin, three engineers, James Truchard, Jeff Kodosky, and Bill Nowlin, have already founded National Instruments in 1976 with a simple focus: connect instruments to computers so that scientists and engineers can get results faster. Early products revolve around GPIB and measurement cards, yet the deeper ambition is to make the computer feel like an instrument that you can compose and recompose at will.

James Truchard, Jeff Kodosky and Bill Nowlin



LabVIEW While Loop abstraction within it's diagram IDE

Jeff Kodosky has a simple question that sounds audacious in that context. **What if programming for measurement and control looked like drawing a circuit that runs?**

LabVIEW is the answer. The front panel is where you place knobs, charts, and indicators. The block diagram is where you wire boxes that do work. Data flows along wires and triggers execution when inputs are ready. The result feels like an oscilloscope pointed at your own program. You click run and watch values ripple through the graph in real time. You correct mistakes by looking, not guessing.

As LabVIEW made dataflow tangible, NI solved the other half of the problem with a unified driver stack: **NI-DAQmx**. Instead of coding per-board quirks, engineers declared *what* they wanted, sample clock, channel list, trigger, buffer size and DAQmx handled *how* to talk to multiplexed ADCs, counters, timers and DMA behind the scenes. Critically, the DAQmx task model mapped cleanly to LabVIEW's block diagram: configure once, start, read/write, stop, with deterministic timing and good diagnostics. That pairing "visual dataflow + a portable hardware abstraction" is the historical proof that orchestration and I/O can live in one mental model. GO HW borrows that lesson: keep the graph as the plan, keep a clean runtime, and expose hardware primitives as first-class nodes instead of ad-hoc glue.

Control is part of the picture from day one. If, For, and While sit beside math nodes and filters. They let you express choices, loops, and orderly repetition with the same visual

clarity. The idea is not to hide complexity. The idea is to make it visible so teams can reason about behavior, timing, and state without reading a wall of text.

Why does this matter to our story about ONNX. Because it proves that graphs are a practical way to build and operate complex systems. It shows that an IDE can help non-specialists work confidently with powerful machinery when the model of computation matches how they think. It also shows that orchestration is not a footnote. It is the method that turns a collection of operations into a working system.

LabVIEW in one minute

- *1986*
- *Pioneered the industrial application of graph computing through LabVIEW Visual dataflow, not syntax rules*
- *Two synchronized views, front panel and block diagram*
- *Live execution, you can watch and debug*
- *Control structures that make behavior explicit*

That same clarity is what we wanted for modern AI workflows, which led to our first toolkit and the lessons that shaped the pivot.

Graiphic chapter 1 — A Keras style toolkit that hit two walls

HAIBAL (2022), Our first LabVIEW deep learning toolkit spoke fluent Keras. It offered layers you could stack, an H5 file you could save, and a clean mental model that many engineers already knew. That choice made adoption easy, yet it hid a mismatch. Keras layers are friendly abstractions, while ONNX and modern runtimes reason in finer grained nodes. PyTorch and TensorFlow can export models as operator level graphs. Our layer centric design could not round trip neatly with that world. Converters had to guess how a stack of layers mapped to a set of low-level ops. Small gaps turned into friction.

The second wall was speed. We executed through the LabVIEW runtime with a light CUDA bridge. It worked and it was ergonomic, but it was not built for the scale and cadence of tensor compute. The hot path did too many small calls. Memory moved more than it should. Kernels could not fuse across the layer boundaries we had chosen. When we compared common models with mainstream frameworks, we saw the gap in latency and throughput.

Both walls taught the same lesson. The file format and the execution engine must sit at the center. An editor that feels good is not enough if the artifact is not native to the ecosystem you target. A runtime that feels integrated is not enough if it cannot plan whole graphs and keep the hot path tight. We needed to keep the ergonomics of layers for engineers, open the door to node level editing for researchers, and anchor the truth in an ONNX file that any tool could read.

What we kept and what we changed

- *Kept the clarity of layers for quick prototyping*
- *Added node level access for custom research work*
- *Replaced H5 as the primary artifact with ONNX as the single source of truth*
- *Moved execution from the LabVIEW runtime to an engine built for graphs*

Graiphic chapter 2 — one file, one engine, one cockpit

Editors change, hardware changes, teams change. The artifact stays the same and the engine keeps it honest.

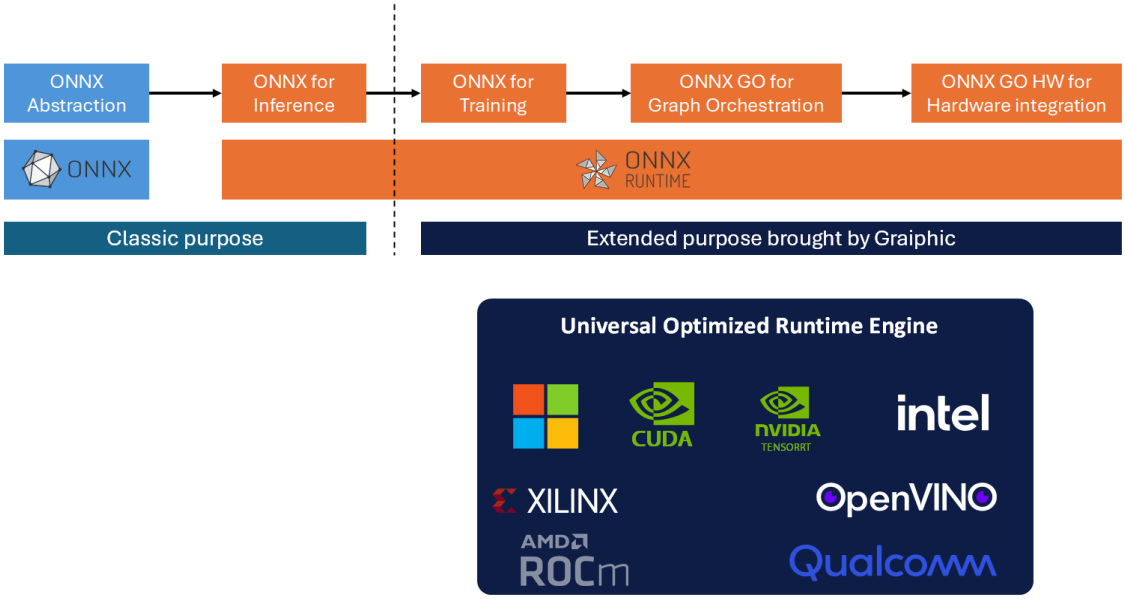
We support two lanes without splitting the road. In layer mode an engineer builds with friendly blocks that feel like Keras. In node mode a researcher edits fine grained operators and custom subgraphs. Both lanes write to the same ONNX graph, with the same shapes, the same weights, the same metadata. You can start in layers for speed, drop to nodes for precision, and never leave the file that ships.

Performance stops being an accident and becomes a property of the build. ONNX Runtime compiles the graph into a session, fuses compatible chains, allocates memory arenas, and plans formats so tensors do not bounce around. You call the session many times with the same inputs, and the hot path stays tight. Latency becomes predictable, throughput scales, energy stops leaking into copies you did not ask for.

The day-to-day experience improves too. The ONNX file is versioned like code. Diffs are meaningful because the graph is declarative. Tests can run on a CPU provider during development and switch to an accelerator provider in staging with the same logical behavior. When something regresses, you inspect the optimized graph and see what changed rather than guess which script drifted.

Graiphic did not approach ONNX as a fixed standard limited to AI inference. From the beginning, we considered ONNX as a foundation for a broader category of graph-based execution systems. This perspective led to a series of structured extensions and contributions that progressively expanded the ONNX Runtime ecosystem.

The first breakthrough was the integration of training workflows directly into ONNX graphs. Through our internal platform SOTA, we demonstrated that neural network training, including backpropagation, could be described and executed within ONNX Runtime. This eliminated the need for Python training loops or external scripting, proving that ONNX could support dynamic learning operations rather than being limited to static inference.



ONNX Evolution: From AI Inference to Full Graph-Based System Orchestration

Building on this foundation, we introduced ONNX GO, a framework for Graph Orchestration. ONNX GO extended the ONNX specification with control flow constructs such as conditional branching (If), iteration (Loop), and structured scans (Scan). These additions allowed ONNX graphs to express general-purpose logic, including decision trees, processing pipelines, and reactive system behaviors.

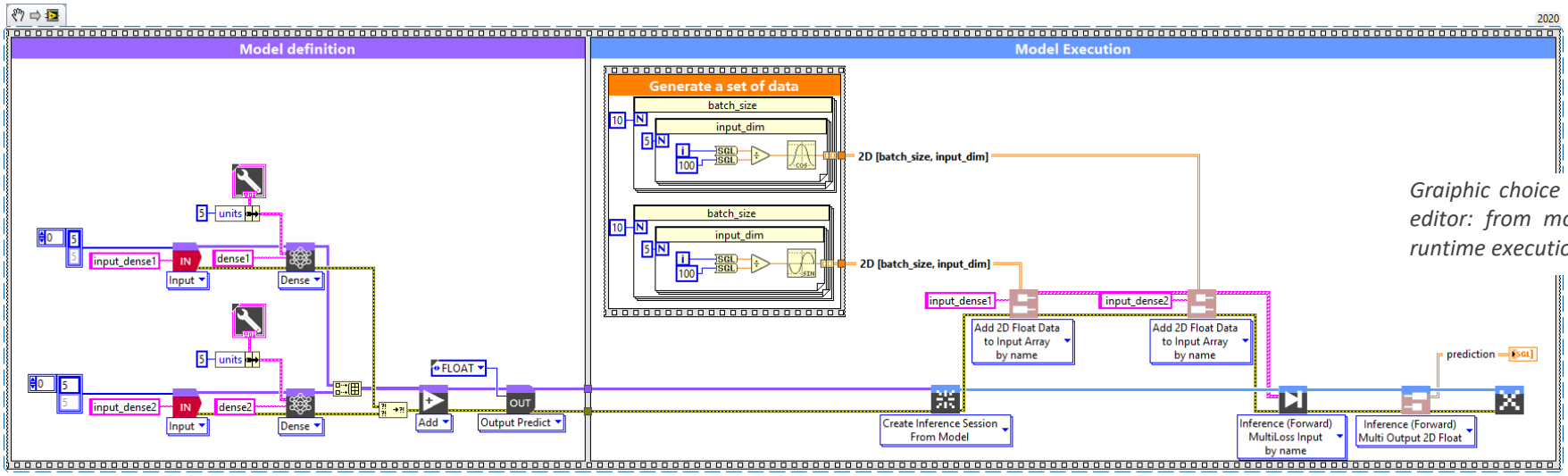
To make these capabilities accessible, we integrated ONNX GO into LabVIEW, creating a visual environment for graph composition and execution. Engineers could now design, modify, and run ONNX-based systems across various platforms using a graphical interface that supports modularity, clarity, and live debugging.

This progression from inference, to training, to full orchestration laid the groundwork for the next step. ONNX GO HW emerged as a natural extension, introducing hardware-level access as a native part of ONNX graphs. It completes the transformation of ONNX into a universal execution layer capable of describing both software logic and physical hardware control in a single, portable format.

A small quality of life loop closes the circle. Import a legacy model, normalize it to a clean ONNX graph, run quick shape checks, auto generate minimal docs from operator metadata, and compile a warm session for your target. The file becomes the contract. The runtime becomes the guarantee.

Rules of engagement

- *One ONNX bundle is the source of truth for models and transforms*
- *Treat layer mode as a convenience, not a trap*
- *Keep pre and post processing in the graph when possible*
- *Inspect the optimized graph like you review code*
- *Compile once per target for stable memory and timing*

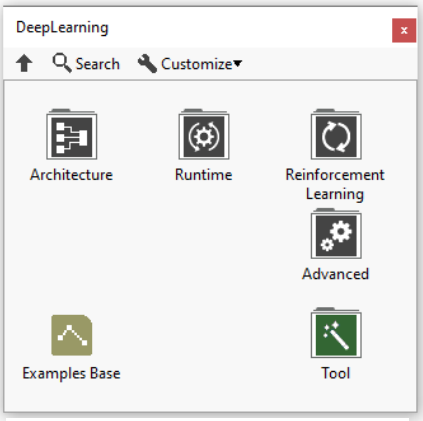


Graiphic choice - LabVIEW ONNX editor: from model definition to runtime execution (Diagram view)

The editor reads left to right. In the purple area “Model definition” you build the ONNX graph with blocks (Inputs, Dense, Add, Output) and keep a clean ONNX artifact. In the blue area “Model Execution” you open an ONNX Runtime session from that model, inject batched inputs, run the forward pass, and collect the outputs. The same ONNX file drives both validation and execution; only the view changes from authoring to running.

Graphs made complex systems visible. The next step is to let the same graph do more than predict. From inference to training and orchestration, it is still a graph.

From inference to training and orchestration, still a graph



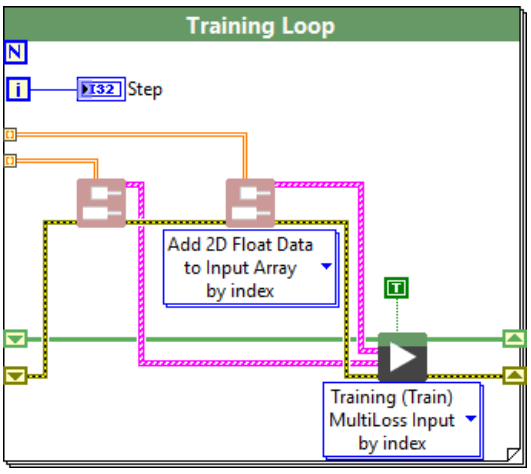
LabVIEW ONNX editor: Palette

We keep one artifact, the ONNX graph, and we teach it new moves. Conceptually a training graph adds a loss, gradients, and an optimizer to the forward path. These pieces fit the ONNX mindset and keep the artifact versionable and auditable. They make training a plan you can read instead of a pile of scripts.

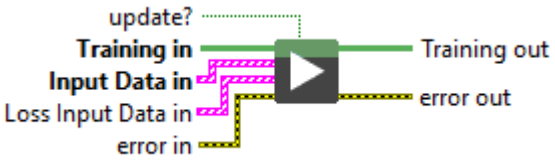
Here is the practical nuance in our current stack. Today the **orchestration loop lives in LabVIEW**, not inside the ONNX graph. LabVIEW plays the role of If, Loop, and Scan in its own dataflow. We tick the loop, feed inputs to the graph, run one forward pass, collect outputs, collect outputs, and repeat. This keeps high level control familiar and debuggable while we gradually move schedule logic into ONNX when it is mature enough. The diagrams you shared show this clearly. The model is defined once, a session is created, and LabVIEW drives the sequence one inference at a time.

We support three session flavors in ONNX Runtime so teams choose the right granularity without changing tools:

- **Inference session** Classic forward only. Use for serving and evaluation.
- **Training session** in fit mode (green wires) Forward, loss, backward, and update handled as a single callable.
- **Academic session** Forward and backward are exposed separately so you can inspect tensors, plug custom losses, or prototype research ideas.



LabVIEW Diagram Orchestration of ONNX Runtime Training Inference



LabVIEW Training Inference functionality

This hybrid phase is intentional. It delivers value now and sets a clean path to full in graph orchestration later. You already get compiled sessions, fused kernels, and stable memory on each target. You already keep preprocessing and metrics close to the model so runs are reproducible. You already ship a single ONNX file that moves from experiment to evaluation to serving. What changes next is where the schedule lives. We will gradually

encode epoch loops, mini batch steps, and early stopping with ONNX control flow so the artifact carries both computation and cadence.

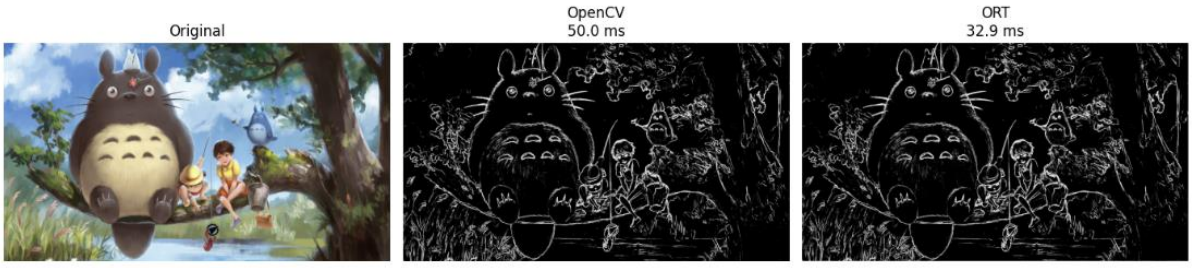
Micro callout

- *Today: LabVIEW owns the loop and calls ONNX Runtime each tick*
- *Tomorrow: control flow migrates into ONNX using If, Loop, and Scan*
- *Always: one ONNX file, one compiled session per target, the same truth in every phase*

The NI Connect moment

We arrived at NI Connect with one story to tell. A clean LabVIEW experience on top of ONNX and ONNX Runtime for deep learning, with the orchestration loop living in LabVIEW. The first discussion with NI engineers changed the scope in the best possible way. If the graph can express complex deep learning, it can also express simpler building blocks from the LabVIEW palette. That idea kicked off the Accelerator Toolkit. The goal was straightforward. Generalize ONNX beyond deep learning and use ONNX Runtime to execute any compute graph efficiently.

Results followed quickly. A matrix multiplication benchmark on CPU showed the Accelerator beating native LabVIEW by a wide margin. At size 8000 the time ratio reached about 5.5 in our test VI, with ten iterations per size for fair timing. The same pattern appeared in computer vision. A Sobel edge detector built as an ONNX graph and run with ONNX Runtime outpaced an OpenCV implementation by roughly 30 to 40 percent depending on resolution. These two measurements gave us confidence that the generalized graph route was sound. The videos and screenshots we shared with NI captured the effect clearly.



Comparison between OpenCV and ONNX runtime time execution performance on Sobel Edge Detector on an CPU execution provider (ORT : ONNX Runtime)

The second moment came the next day with an NI engineer who had missed the first meeting. Your idea to generalize graphs is good, he said, but how do you control hardware signals with this technology. The question landed and stayed. It reframed the problem from pure acceleration to timing and alignment with the real world.

The timeline matters. In May we had only the deep learning toolkit and a LabVIEW driven loop that fed the graph and called ONNX Runtime step by step. In July we shipped the Accelerator Toolkit to prove that generalized graphs run fast for pre and post processing and for standalone math. In August we began shaping the hardware path. The order is

deliberate. Show speed first, then bring timing into focus, then extend the model to the physical layer. Step by step.

Two ideas were born in those conversations with NI. First, treat ONNX as a general graph that can execute efficiently beyond deep learning. Second, answer the hardware question with a design that makes timing and control as explicit as the math. The first idea is already in the product. The second is the seed we are growing now.

Generalizing the graph is only half the story. To act on the physical world, we need to speak the language of chips. A short tour of SoCs makes the stakes concrete.

The Origin of SOTA: Why This Ecosystem Had to Exist

SOTA did not appear by accident. It was born from years of observing the same problem repeat across companies, laboratories, universities, and engineering teams. Everywhere, the story was identical. Artificial intelligence was advancing rapidly, yet the tools required to build AI systems were scattering in every direction.

Every project required a different combination of frameworks, languages, runtimes, annotation tools, training utilities, hardware libraries, and deployment pipelines. Nothing worked the same way twice. Nothing was designed to be unified. Each step required a new tool, a new dependency, a new layer of complexity.

This fragmentation created a paradox.

- AI was becoming more powerful and more intelligent, yet the process of creating
- AI remained heavy, slow, and inaccessible.
- The real limitation was not the models.
- The real limitation was the ecosystem surrounding them.

As engineers, we experienced this pain directly. Every new idea demanded hours of setup, configuration, debugging, and rewriting. Every customer needed custom scripts just to integrate a model into their architecture. Every researcher had to switch between multiple environments to try a simple experiment. The potential of AI was being held back by the tools meant to enable it.

From this frustration emerged a clear insight.

- AI needed a foundation, not another isolated tool.
- A foundation that could unify the entire lifecycle.
- A foundation that would make AI development enjoyable instead of exhausting.
- A foundation that engineers could trust, that researchers could explore, and that industry could deploy at scale.

This foundation had to meet three essential conditions.

- It had to be **accessible**, so that any engineer could design and deploy AI without mastering a dozen programming paradigms.
- It had to be **interoperable**, so that models trained anywhere could run everywhere.
- It had to be **graphical**, because the best way to understand and orchestrate complex systems is through visual and intuitive representations.

These three principles guided the creation of SOTA.
We chose LabVIEW for its graphical power, industrial maturity, and unmatched speed for prototyping.
We chose ONNX Runtime for its universality, performance, and ability to run the same model on any hardware.
We built toolkits, libraries, and automation layers that transform AI workflows into a fluid and coherent experience.

Bit by bit, SOTA became more than a toolkit. It became an ecosystem. It became a platform where engineers can build neural networks with clarity. Where researchers can experiment with full control. Where companies can deploy AI on GPUs, ARM devices, FPGAs, and industrial controllers with confidence. It became the technological base that the AI world was missing.

SOTA exists because AI needed a unified home.

- A place where innovation feels natural.
- A place where complexity becomes elegant.
- A place where intelligence is not just executed, but orchestrated.

This is the origin of SOTA.

- A response to fragmentation.
- A commitment to accessibility.
- A belief that AI should be powerful, intuitive, and universal.

The Problem: Fragmentation, Complexity, and the Missing Foundation

Before SOTA, the journey to create an AI system was anything but straightforward. Each project looked like a puzzle assembled from unrelated pieces. A model might be trained in one framework, exported through another tool, optimized by a different library, deployed on a custom runtime, and finally integrated into an industrial architecture written in a completely separate language. Nothing was unified. Nothing was simple. And nothing was designed to last.

This fragmentation produced several critical barriers.

- First, **complexity grew faster than capability**.
  Engineers were forced to switch constantly between Python scripts, annotation platforms, GPU toolkits, cloud dashboards, and deployment pipelines. Each tool carried its own learning curve, its own configuration rules, its own version constraints. Even experts found themselves navigating endless incompatibilities instead of focusing on innovation.
- Second, **skills became a bottleneck**.
  To bring a single model from idea to production, teams needed a data scientist, a software architect, a hardware specialist, and sometimes an additional engineer for integration. The cost was enormous. The process was slow. And for many organizations, the expertise simply did not exist.
- Third, **portability was fragile**.
  A model trained in PyTorch might not behave identically once exported. A model optimized for GPU servers often failed on embedded hardware. Each environment required adjustments, scripts, and patches, creating a fragile chain that was difficult to maintain and nearly impossible to industrialize.
- Fourth, **graphical understanding was missing**.
  AI workflows are inherently complex. They involve data flows, tensor operations, backpropagation pipelines, and execution graphs. Yet the tools available forced users into text-centric workflows that hid the system's structure instead of revealing it. Engineers could not "see" their model. Researchers could not visually debug their logic. Industry could not easily validate how the AI behaved inside a larger architecture.
- Finally, **no single tool covered the full lifecycle**.
  Annotation tools were separated from training environments. Training environments were isolated from deployment pipelines. Deployment pipelines were disconnected from hardware acceleration. And generative models required yet another set of frameworks entirely.

The result was predictable.

- Innovation slowed down.
- Costs increased.
- Maintenance became a challenge.
- Portability broke regularly.
- And companies abandoned projects because the ecosystem itself was too difficult to navigate.

AI had intelligence. But it lacked the structure.

The industry needed a foundation. A foundation that would unify, simplify, accelerate, and illuminate every step of the AI lifecycle. A foundation able to erase fragmentation and create a coherent experience from raw data to industrial deployment.

This missing foundation is exactly what SOTA set out to provide.

Why SOTA Changes Everything

SOTA transforms the way we build artificial intelligence because it solves the root problems that have limited AI adoption for years. Instead of adding another tool to an already crowded landscape, SOTA reshapes the entire experience of creating and deploying AI systems. It introduces a new way of thinking, a new way of building, and a new way of orchestrating intelligence.

- The first breakthrough is **unification**.
  SOTA replaces a scattered collection of disconnected tools with a single, coherent environment. Data preparation, annotation, training, optimization, deployment, and execution all live inside one ecosystem. The friction disappears. The learning curve becomes predictable. AI development becomes linear, understandable, and enjoyable. For the first time, engineers, researchers, and industry teams can collaborate inside the same platform without losing context or compatibility.
- The second breakthrough is **graphical understanding**.
  SOTA uses visual language that reveals the flow of intelligence. Neural networks become structures you can see and manipulate. Training pipelines become intuitive. Hardware orchestration becomes transparent. Mistakes become easier to diagnose, and solutions become faster to implement. Instead of deciphering layers of code, users interact directly with the logic that drives the model. This unlocks a new level of clarity and accessibility, removing barriers that previously slowed down experimentation and innovation.
- The third breakthrough is **interoperability powered by ONNX Runtime**.
  SOTA does not lock users into a proprietary ecosystem. It embraces the industry's most open and universal AI standard. Models created anywhere can be imported instantly. Models trained in SOTA can run on any device, from GPU servers to ARM processors and FPGAs. This eliminates the constant need to rewrite code, convert frameworks, or adjust architectures for specific hardware. The result is complete freedom in choosing tools, technologies, and deployment environments.
- The fourth breakthrough is **automation and acceleration**.
  SOTA integrates optimizations, GPU acceleration, and automated training pipelines directly into its core. What once required specialized knowledge becomes accessible to any engineer. What once required manual configuration becomes automatic. SOTA reduces prototyping time dramatically and enables rapid iteration on models, datasets, and architectures. The impact is immediate: shorter development cycles, lower costs, and more innovative projects brought to life.
- The fifth breakthrough is **industry readiness**.

SOTA is not an academic prototype. It is built for real industrial systems. It integrates naturally with embedded devices, real-time controllers, vision systems, and automation architectures. It supports deployment in environments where stability, reliability, and performance are non-negotiable. SOTA brings AI out of notebooks and research labs and places it directly into machines, factories, laboratories, and products.

- The sixth breakthrough, and the most transformative, is **Graph Orchestration**. For the first time, AI does not stop at inference. With SOTA, ONNX becomes more than a way to execute models. It becomes an active participant in the architecture of the system. Thanks to LabVIEW and its graphical dataflow paradigm, SOTA introduces a completely new capability: orchestrating neural networks, processing pipelines, business logic, memory flows, and hardware interactions within a single unified graph.

  This is a new technological frontier. It allows AI models to live inside complex architectures instead of isolated black boxes. It enables deterministic execution, predictable timing, visual debugging, and seamless integration with state machines, actors, real-time systems, and distributed hardware. This exclusive combination of LabVIEW and ONNX creates advantages that did not exist before: unique visibility into model behavior, new levels of reliability, native compatibility with industrial systems, and a strategic gain in development speed, safety, and performance.

  What once required several languages, tools, and glue code can now be expressed in a single orchestrated graph.

- The final breakthrough is **future alignment**.
  SOTA is not only a solution for today. It is the technological base for GO HW, GO GenAI, and GO IML. Through these upcoming technologies, SOTA prepares the transformation of ONNX from a static model format into a dynamic graph capable of orchestrating intelligence, logic, hardware, and energy. This evolution marks the beginning of a new era where AI systems become fully integrated, interpretable, and universally deployable.

In short, SOTA changes everything because it makes AI development simple, visual, universal, accelerated, orchestrated, and ready for the real world.

- It turns complexity into clarity.
- It turns fragmentation into harmony.
- It turns potential into performance.

And through Graph Orchestration, it turns intelligence into a living system.

SOTA is not just another tool. SOTA is the platform that finally allows AI to flourish without friction, without barriers, and without limits.

The Vision: A Unified Graphical Environment for AI

The vision behind SOTA emerged from a simple observation. Artificial intelligence had become powerful, essential, and full of promise, yet the tools used to build AI systems remained scattered, fragile, and excessively technical. Teams were forced to combine multiple frameworks, scripts, runtimes, and utilities, each with its own requirements and syntax. Innovation slowed down not because of a lack of ideas, but because the ecosystem made those ideas hard to express.

SOTA was created to unlock this potential by redefining the way AI is developed.

The core vision is to offer **a unified graphical environment** that allows anyone to build intelligent systems without the usual friction of traditional programming. Instead of requiring mastery of multiple syntaxes, SOTA focuses on what truly matters: understanding the problem, designing the solution, and deploying it efficiently. In this environment, deep learning, computer vision, generative models, GPU acceleration, data annotation, model importation, and industrial deployment all coexist within one coherent ecosystem.

This unified approach transforms the experience of AI development. SOTA provides a complete suite of **intercompatible tools and toolkits**, each designed to work seamlessly with the others. Whether the user is training a neural network, annotating a dataset, optimizing GPU operations, importing an ONNX model, deploying on embedded hardware, or integrating AI into a real-time architecture, every task takes place inside the same visual and intuitive framework.

This clarity brings something new and exclusive to the AI world.

For the first time, ONNX models can be orchestrated and integrated graphically inside complex system architectures. Because SOTA relies on LabVIEW's graphical dataflow model, AI does not live in isolation anymore. It becomes a functional component inside a larger system, with predictable timing, visual execution, and full interoperability with logic, sensor data, control systems, and high-level software structures. This is the essence of **Graph Orchestration**, a key innovation that elevates AI beyond inference and into system-wide intelligence.

But the vision of SOTA goes even further.

It challenges a misconception that has slowed down countless projects: the idea that software syntax defines technical expertise. Knowing how to write Python or C++ code does not automatically provide mastery in AI, robotics, embedded systems, software architecture, or scientific reasoning. These fields require deep domain knowledge that often gets overshadowed by the complexity of coding itself.

SOTA reverses this dynamic.

By simplifying the hardest parts of software development and removing syntax barriers, SOTA enables engineers, researchers, and domain experts to focus on **what they know best**: designing smarter algorithms, experimenting with new ideas, refining architectures, solving practical problems, and accelerating innovation. The environment becomes a catalyst rather than an obstacle.

The result is a new development paradigm where accessibility does not compromise performance. SOTA guarantees optimized execution through ONNX Runtime, reliable processes through its graphical architecture, and industrial-quality deployment across GPUs, ARM platforms, FPGAs, and real-time controllers. It combines the ease of visual design with the power of highly optimized backend execution.

This vision creates a world where AI is no longer reserved for those who can navigate complex syntax or assemble fragmented pipelines. AI becomes a domain where creativity flows naturally, where ideas become prototypes quickly, and where prototypes become real industrial systems with confidence and precision.

SOTA embodies this vision.

A unified, graphical, interoperable, and accessible environment that turns intelligence into something that can be designed, visualized, orchestrated, and deployed seamlessly.

It is the beginning of a new way to build AI. A way where innovation takes the lead, and technology finally works in harmony with human expertise.

Components and Capabilities of SOTA

SOTA is a complete and coherent ecosystem built from toolkits and tools that all work together to simplify, accelerate, and unify the entire lifecycle of artificial intelligence. Each component has been designed with a clear goal: deliver powerful capabilities through a graphical environment that removes unnecessary complexity while preserving performance, flexibility, and industrial reliability.

The strength of SOTA lies not only in the individual toolkits but in the way they interact. Every component is intercompatible. Every workflow is continuous. Every feature contributes to an environment where AI development becomes intuitive, efficient, and ready for deployment.

SOTA includes four main toolkits and several strategic tools.

Deep Learning Toolkit

The Deep Learning Toolkit is the heart of SOTA. It provides a graphical environment for designing, training, analyzing, and deploying neural networks using ONNX Runtime as its execution backbone and LabVIEW graphical language as its GUI.

Its capabilities include:

- graphical construction of neural networks
- support for engineer-level abstraction and research-level granularity
- customizable training pipelines through visual flows
- integration of advanced reinforcement learning algorithms
- optimized execution on CPU, GPU, ARM, and FPGA through ONNX Runtime
- complete compatibility with models created in PyTorch, TensorFlow, Keras, or exported through FIG

The toolkit allows users to design deep learning architectures visually, eliminating the complexity of syntax while giving full control over structure and behavior. Training becomes transparent, interpretable, and efficient.

Combined with SOTA's orchestration capabilities, the Deep Learning Toolkit becomes a powerful engine for building intelligent systems integrated directly into real-world applications.

Computer Vision Toolkit

The Computer Vision Toolkit extends SOTA into all vision-driven applications. It provides tools for:

- real-time image and video processing
- object detection, classification, and segmentation
- preprocessing and augmentation functions
- integration with hardware vision systems and embedded devices
- preparation of pipelines for training and fine-tuning vision models

This toolkit makes computer vision accessible even to users who do not specialize in AI. By connecting the CV Toolkit with the Deep Learning Toolkit, SOTA enables complete end-to-end workflows: from dataset creation to model deployment, all within one graphical environment.

Accelerator Toolkit

The Accelerator Toolkit is not a simple collection of GPU functions. It represents a new technological paradigm inside SOTA. For the first time, an AI company has extended the idea of ONNX nodes far beyond neural network inference to create a **general-purpose acceleration engine for any LabVIEW computation**.

Instead of treating ONNX Runtime as a tool reserved for executing AI models, Graiphic reimagined it as a universal backend capable of accelerating entire LabVIEW graphs. The Accelerator Toolkit acts as an editor and executor of graphical palettes, powered by the full suite of ONNX Runtime Execution Providers such as CUDA, TensorRT, OneDNN, OpenVINO, and DirectML.

This means that tensor transformations, image operations, numerical computations, preprocessing, postprocessing, and high-cost calculations of any kind can now run on GPU, NPU, or optimized CPU backends with industrial reliability.

This innovation is unique in the world.

Graiphic is the first company to generalize ONNX nodes for purposes other than AI inference. By transforming every operation into an accelerated ONNX-compatible node, the Accelerator Toolkit unlocks a new category of performance: the ability to optimize **any** computation, not just deep learning layers.

This required an exceptional level of engineering. We created the most exhaustive public cartography of ONNX Runtime capabilities by testing every node across all execution providers, available here: https://github.com/Graiphic/ONNX-Runtime-Execution-Providers-Tester

This work allowed us to understand how each provider behaves, which operations it accelerates, and how to build a unified runtime strategy tuned for maximum performance inside LabVIEW.

The result is a breakthrough.

Heavy operations that traditionally slowed down vision systems, LLM pipelines, scientific calculations, and real-time applications can now be transformed into fast, highly optimized GPU workflows. The synergy with the Deep Learning Toolkit creates a continuous chain of acceleration, from preprocessing to inference to postprocessing, all inside a single graphical environment.

The Accelerator Toolkit delivers performance levels comparable to modern Python ecosystems while providing something they cannot offer: **native graphical integration, deterministic dataflow, and unified orchestration inside LabVIEW.**

It is not just an accelerator.

It is the engine that elevates SOTA from an AI toolkit to **a full system acceleration platform**, capable of optimizing entire architecture and enabling a new generation of intelligent, high-performance, and visually orchestrated applications.

GenAI Toolkit

The GenAI Toolkit brings large language models and multimodal intelligence into the SOTA ecosystem, making advanced generative AI available inside LabVIEW's graphical environment. Today, the toolkit relies on LlamaCPP, a high-performance inference engine that allows users to run large models locally with excellent efficiency. This gives engineers and researchers immediate access to modern LLMs without cloud services or complex infrastructure.

However, the future of generative AI inside SOTA is already emerging from the Deep Learning Toolkit. Because this toolkit is fully based on ONNX and ONNX Runtime, it has already demonstrated the ability to execute models such as Llama 3.1 and the multimodal model Florence 2. These successful executions prove something essential. Generative AI does not need to remain tied to specialized libraries or custom runtimes. It can become part of a unified ONNX architecture, accelerated by the same execution providers that power all other AI operations in SOTA.

This achievement defines the direction of the GenAI Toolkit. The long-term vision is to move from LlamaCPP to a complete ONNX and ONNX Runtime foundation. This evolution is the core of GO GenAI, the component of SOTA that will unify generative intelligence with the entire ecosystem.

GO GenAI will allow large language models and vision language models to act as native ONNX components. Their layers will be executed by ONNX Runtime Execution Providers such as CUDA, TensorRT, OneDNN, OpenVINO and DirectML. This approach will accelerate transformer architecture, simplify deployment across heterogeneous hardware, and enable full interoperability with the Deep Learning Toolkit and the Accelerator Toolkit.

Most importantly, GO GenAI introduces Graph Orchestration for generative intelligence. Models become nodes inside a graphical dataflow rather than isolated engines. They interact with logic, sensors, memory structures, computer vision pipelines, reinforcement learning components and real-time processes. This transforms generative AI from a separate capability into a fully orchestrated and visual part of complex architectures.

This is a major shift compared to traditional GenAI ecosystems. Most existing solutions depend on Python APIs or cloud endpoints, which keeps generative models isolated from the rest of the system. SOTA follows a different path. It brings generative intelligence into the same graphical, deterministic and hardware-aware environment as all other AI and numerical operations.

Thanks to this integration, the future GenAI Toolkit will offer:

visual creation of multimodal pipelines using LLMs, VLMs and signal or vision data predictable and explainable orchestration of generative processes hardware acceleration through ONNX Runtime provider selection graphical debugging of generative logic industrial deployment across embedded, real-time and GPU platforms native compatibility with the Deep Learning Toolkit and the Accelerator Toolkit

The Deep Learning Toolkit already proves that this vision is technically sound. Running Llama 3.1 and Florence 2 inside ONNX shows that generative AI can be unified

with the entire SOTA ecosystem. The GenAI Toolkit will build on this foundation and turn it into a complete, powerful and fully orchestrated solution.

SOTA is the first platform to extend generative AI into a true graphical system architecture. It is not only adding GenAI to LabVIEW. It is redefining how generative intelligence is executed, accelerated, integrated and orchestrated in real systems.

Tools Integrated into the SOTA Ecosystem

In addition to its toolkits, SOTA includes several tools that complete the ecosystem and make it practical, intuitive, and ready for deployment.

FIG (File Importer and Generator)

FIG is the interoperability gateway of SOTA. It allows users to:

- import models from PyTorch, TensorFlow, Keras to ONNX
- visualize model architectures
- convert models for execution inside SOTA
- inspect structures at different levels of detail

This tool ensures complete freedom of workflow. Users can train models anywhere and deploy them instantly in SOTA without rewriting code.

Annotation Tool

The Annotation Tool is a complete solution for computer vision dataset creation and training automation. It supports:

- dataset import from Roboflow (200 million images, 42 formats)
- manual and semi-automatic annotation
- data augmentation
- dataset editing and versioning
- launching training pipelines directly through the Deep Learning Toolkit

For the first time, users can go from raw images to a fully trained model without leaving the SOTA environment. This dramatically increases productivity and reduces fragmentation in vision projects.

Builder Tool

The Builder Tool is not simply a deployment utility. It is an intelligent assistant that guides users through the creation of complete and reliable LabVIEW distributions. Instead of manually assembling files, configuring paths or adjusting project settings, users interact with a wizard that builds the entire distribution step by step and ensures that the final executable functions exactly as intended.

Builder helps the user create a full deployment package by:

guiding the configuration of project settings ensuring that ONNX models, dependencies and required toolkits are correctly included assisting with the preparation of executables

and runtime files verifying that hardware acceleration providers and system options are properly integrated producing a structured and ready-to-run distribution for industrial use

The result is a smooth and controlled transition from development to operational deployment.
Users no longer need to worry about missing files, incorrect links or misconfigured runtimes. Builders generate a clean and functional distribution that can be deployed immediately on production systems.

By transforming deployment into a guided and reliable process, the Builder Tool closes one of the most complex steps in the lifecycle of an AI application. It guarantees that projects created inside SOTA can be delivered as stable, professional and fully operational executables.

A Unified and Seamless Ecosystem

All these components share the same principles:

intercompatibility, simplicity, performance, and graphical clarity.

SOTA is not a set of isolated products. It is an ecosystem where toolkits and tools reinforce one another, creating a natural environment for:

- rapid prototyping
- industrial deployment
- research experimentation
- educational use
- complex system integration

This unified graphical ecosystem eliminates fragmentation, reduces development time, and empowers engineers, researchers, and industry specialists to focus on expertise rather than syntax.

SOTA provides something truly new:

the ability to design, train, orchestrate, and deploy AI systems in a single visual environment that is simple to learn, powerful to use, and ready for the real world.

Call for Funding: Why Industry Should Invest in SOTA

Graiphic has built the first end-to-end ecosystem where AI, logic, and hardware orchestration live inside a single ONNX graph.

We are now opening a Call for Funding to accelerate the roadmap of SOTA.

Why Invest?

- Strategic Advantage: GO IML creates AI systems that learn faster, generalize better, and deliver reliable results in real-world conditions.
- Portability & Standards: The entire technology is built on ONNX, ensuring seamless deployment across CPUs, GPUs, FPGAs, NPUs, and future hardware ecosystems.
- Energy & Efficiency: Informed learning cuts training time, reduces data requirements, and accelerates the path from prototype to operational deployment.
- Market Reach: GO IML addresses multiple high-impact sectors including robotics, aerospace, defense, industrial automation, and intelligent infrastructure.

What We Offer

- Co-development opportunities with our engineering team.
- Early integration of your platforms and SDKs into GO HW.
- Joint visibility in international standardization efforts (ONNX, DARPA, Horizon Europe, ADRA).
- Shared benchmarking and open-source dissemination to establish your technology as a leader in AI orchestration.

How to Engage

Graiphic is actively seeking:

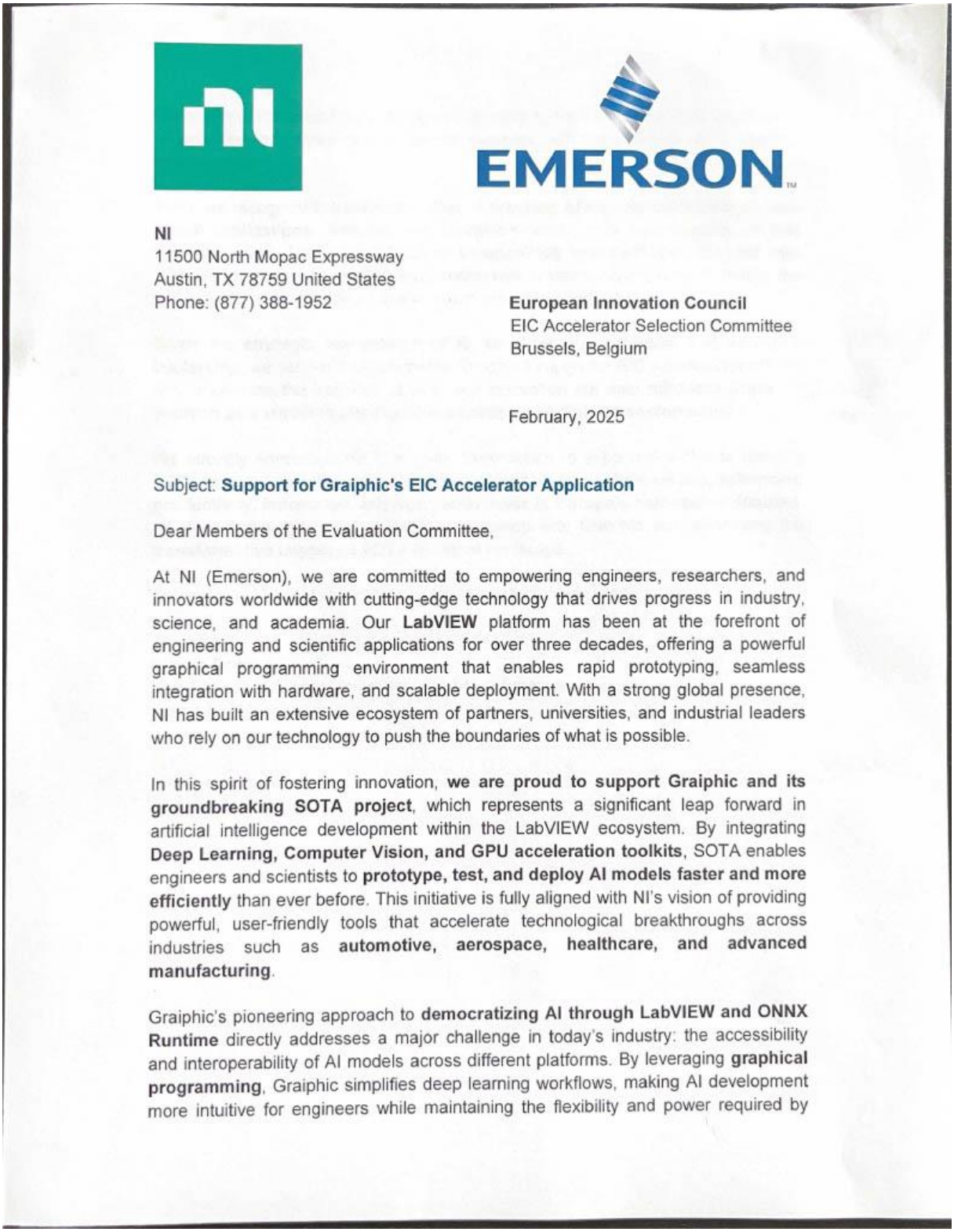
- Equity investors ready to support our growth.
- Industrial sponsors willing to co-fund R&D and test benches.
- Strategic partners (hardware vendors, system integrators, large OEMs) who want their platforms at the heart of the future ONNX Hardware ecosystem.

Join us in shaping the universal cockpit for AI.

Contact: funding@graiphic.io | www.graiphic.io

Annexes

Support Letters



NI
11500 North Mopac Expressway
Austin, TX 78759 United States
Phone: (877) 388-1952

European Innovation Council
EIC Accelerator Selection Committee
Brussels, Belgium

February, 2025

Subject: **Support for Graiphic's EIC Accelerator Application**

Dear Members of the Evaluation Committee,

At NI (Emerson), we are committed to empowering engineers, researchers, and innovators worldwide with cutting-edge technology that drives progress in industry, science, and academia. Our **LabVIEW** platform has been at the forefront of engineering and scientific applications for over three decades, offering a powerful graphical programming environment that enables rapid prototyping, seamless integration with hardware, and scalable deployment. With a strong global presence, NI has built an extensive ecosystem of partners, universities, and industrial leaders who rely on our technology to push the boundaries of what is possible.

In this spirit of fostering innovation, **we are proud to support Graiphic and its groundbreaking SOTA project**, which represents a significant leap forward in artificial intelligence development within the LabVIEW ecosystem. By integrating **Deep Learning, Computer Vision, and GPU acceleration toolkits**, SOTA enables engineers and scientists to **prototype, test, and deploy AI models faster and more efficiently** than ever before. This initiative is fully aligned with NI's vision of providing powerful, user-friendly tools that accelerate technological breakthroughs across industries such as **automotive, aerospace, healthcare, and advanced manufacturing**.

Graiphic's pioneering approach to **democratizing AI through LabVIEW and ONNX Runtime** directly addresses a major challenge in today's industry: the accessibility and interoperability of AI models across different platforms. By leveraging **graphical programming**, Graiphic simplifies deep learning workflows, making AI development more intuitive for engineers while maintaining the flexibility and power required by

researchers. This paradigm shift has the potential to transform **how AI is developed, tested, and integrated into industrial systems, IoT applications, and research environments**.

At NI, we recognize the immense value of **bringing AI capabilities closer to real-world applications**, and we see Graiphic's work as a **key enabler of this transformation**. Their strong focus on **compatibility with LabVIEW, coupled with high-performance AI deployment**, represents a unique opportunity to bridge the gap between cutting-edge AI research and industrial applications.

Given the **strategic importance of AI for Europe's industrial and scientific leadership**, we believe that supporting Graiphic through the **EIC Accelerator** will not only accelerate the adoption of AI-driven innovation but also **reinforce Europe's position as a leader in intelligent automation and digital transformation**.

We strongly encourage the European Commission to support Graiphic in bringing SOTA to its full potential, as its impact will extend across multiple sectors, **enhancing productivity, innovation, and competitiveness in Europe's high-tech industries**. NI looks forward to continuing its collaboration with Graiphic and witnessing the **transformative impact of SOTA** on the AI landscape.

Please do not hesitate to contact us for any further information.

Sincerely,
Norma Dorst
Vice President, Global Marketing | NI T&M at Emerson
norma.dorst@emerson.com