🎮 OmniForge

A Real-Time Game Upscaling Framework

Inject into games to **render** at lower resolutions and upscale with a **hybrid AI+FSR pipeline. Boost FPS** while **maintaining visual quality**.

v0.1 (Alpha) - C++/Vulkan Production Build

The Gamer's Dilemma: Performance or Quality?

SMOOTH GAMEPLAY

1920x1080 | **60 FPS**

STUNNING VISUALS

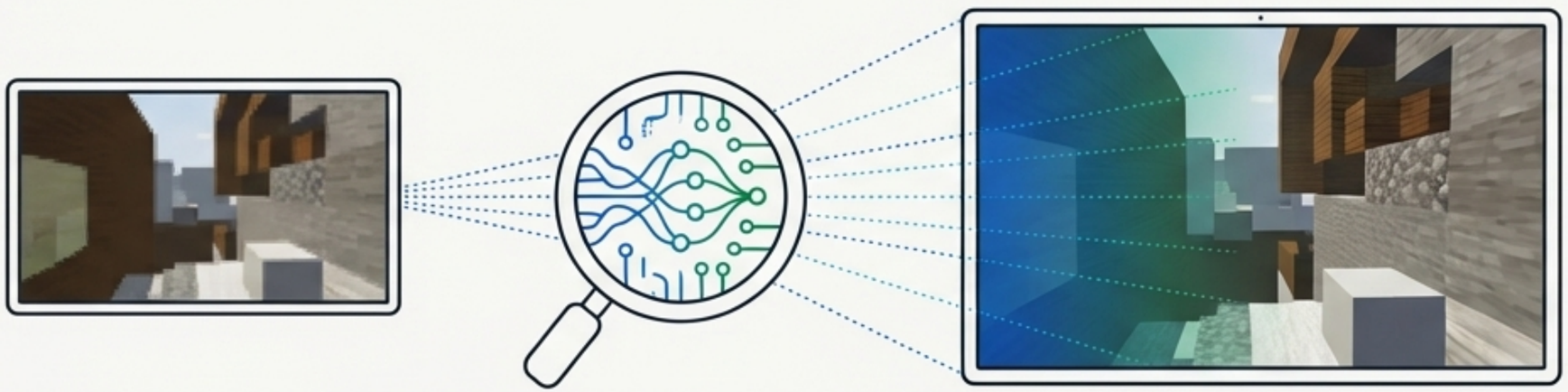
3840x2160 (4K) | **30 FPS**

Running a game at 4K requires **4x more GPU power** than 1080p.
Gamers are forced to choose between high frame rates and high resolution.

The Solution: OmniForge Lets You Have Both

OmniForge is a real-time framework that intercepts game frames, renders them at a lower resolution for speed, and uses a **hybrid AI pipeline** to display them at your target resolution with minimal quality loss.

Think of it as a “Magic Lens”



Low-Res
Game Screen

OmniForge
Magic Lens

High-Res
Enhanced Image

It makes the game look like it’s on a big screen without losing quality, in real-time, 60 times per second.

Technical editorial

The Result: A 65% FPS Increase

Native

- **Render Resolution:** 1920x1080
- **Total Time:** 25ms
- **Visual Quality:** 100%

40 FPS

OmniForge

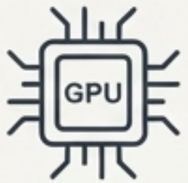
- **Render Resolution:** 960x540 →
- **Total Time:** 15ms (8ms Render + 7ms Upscale)
- **Visual Quality:** ~95%

66 FPS

+65% FPS with minimal quality loss. The game *thinks* it's rendering fast; you *see* a beautiful, high-resolution image.

Scenario: Running a demanding game at 1920x1080 display resolution.

The Complete OmniForge Pipeline



RENDER

The game engine renders at a low resolution (e.g., 960x540).

~8ms (4x faster than native 1080p)

CAPTURE

`omniforge_inject.dll` intercepts the frame via a DXGI hook (`IDXGISwapChain::Present()`) before it reaches the monitor.

<0.1ms

UPSCALE

A hybrid process transforms the low-res frame to high-res. (Detailed next).

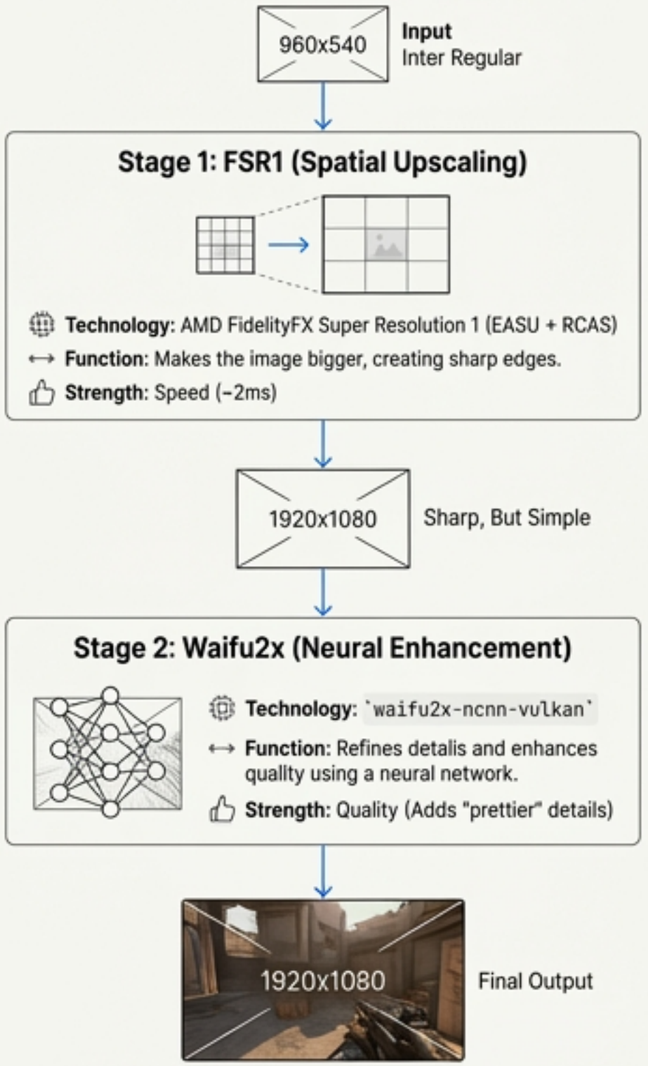
~7ms

DISPLAY

The final, enhanced high-resolution frame is passed to the original `Present()` call and displayed.

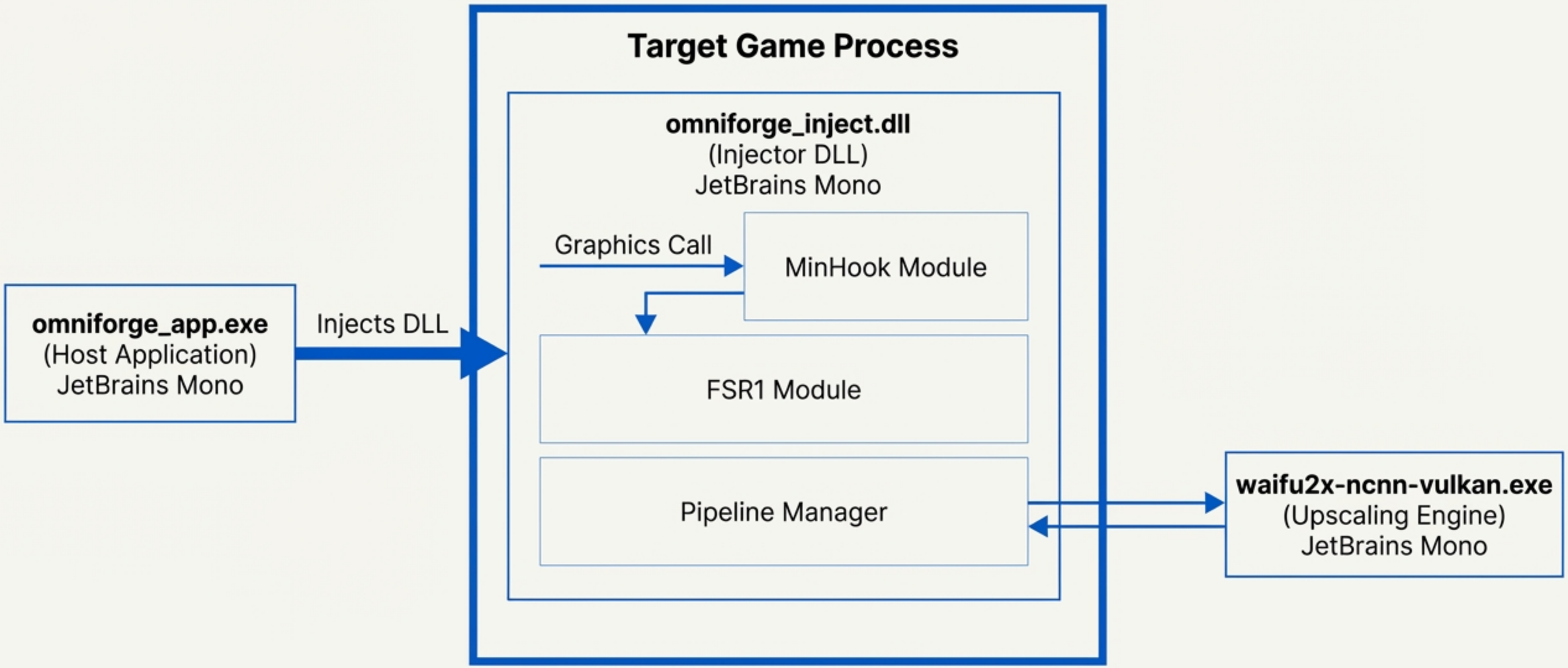
Total time
~15ms = **66 FPS**

Deep Dive: The Hybrid Upscaling Engine



This async/hybrid approach combines the raw speed and sharpness of FSR1 with the intelligent detail enhancement of Waifu2x for an optimal result.

System Architecture: The Blueprint



Component Breakdown

Host Application (`omniforge_app.exe`)

Role: Manages the upscaling session.
Function: A command-line interface (CLI) that injects the DLL into the target game process.

Injector DLL (`omniforge_inject.dll`)

Hooks: Uses `MinHook` (JetBrains Mono) to intercept DirectX `Present()` (JetBrains Mono) calls.
Capture: Extracts the backbuffer texture from the game.
Pipeline: Manages the FSR1 and Waifu2x processing chain.
Metrics: Tracks FPS and frame times for performance monitoring.

Upscaling Engine

FSR1: Implemented via a lightweight header-only library (`EASU/RCAS`) Mono) for spatial upscaling.
Waifu2x: Integrated via a standalone executable (`waifu2x-ncnn-vulkan.exe`) (JetBrains Mono) for high-quality neural processing.

For Gamers: Quick Start Guide

Prerequisites

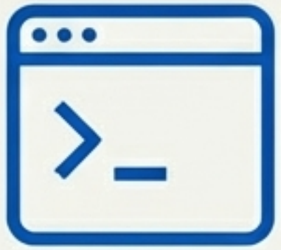
- Windows 10/11 (64-bit)
- Visual C++ Redistributable 2019+ 2019+
- A DirectX 11/12 game

1. DOWNLOAD

Download the latest release from the GitHub repository.

2. UNZIP

Unzip the contents to a folder (e.g., C:\OmniForge).



3. RUN

Open a terminal and run the injector, pointing it to your game's executable.

```
.\bin\omniforge_app.exe --inject
"C:\Path\To\YourGame.exe"
```

For Developers: Build & Contribute

****Requirements****

- Visual Studio 2022 (with C++ Desktop Development workload)
- CMake 3.20+
- Git

Step 1: Clone the Repository

```
git clone https://github.com/Santhoshnadella/omniforge-0.1.git
cd omniforge-0.1
```

Step 2: Build Core Components (Phase 1 & 2)

Builds `omniforge_inject.dll` and `omniforge_app.exe` with FSR1.

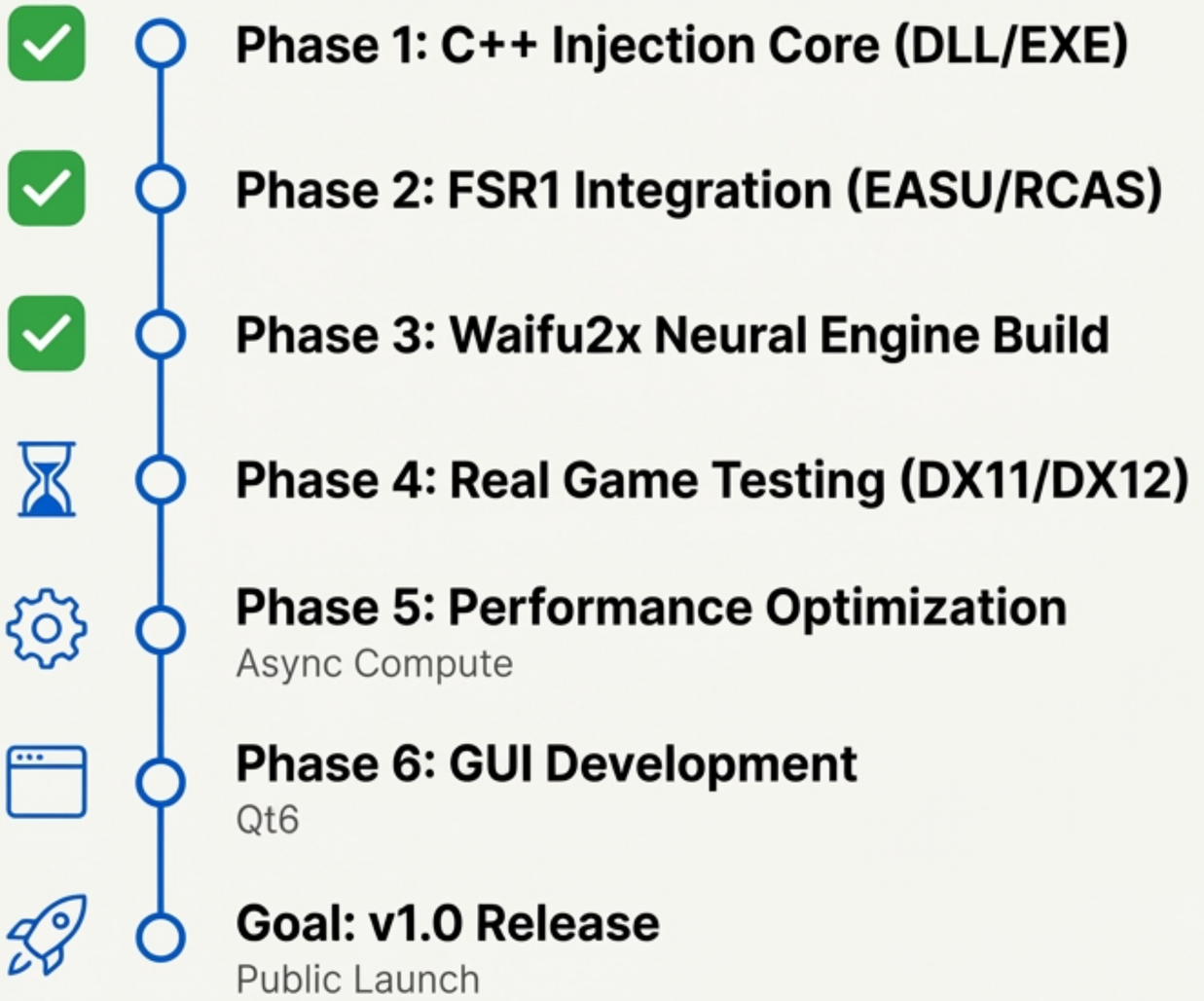
```
.\build_production.bat
```

Step 3: Build Waifu2x (Phase 3)

Builds `waifu2x-ncnn-vulkan.exe` and installs neural models.

```
powershell -ExecutionPolicy Bypass -File .\build_phase3_waifu2x.ps1
```

Project Roadmap: The Path to v1.0



OmniForge Project Vitals

Current Status

- ***Version***: v0.1 (Alpha)
- ***State***: Successfully transitioned from Python proof-of-concept to compiled C++ application.

Technology Stack

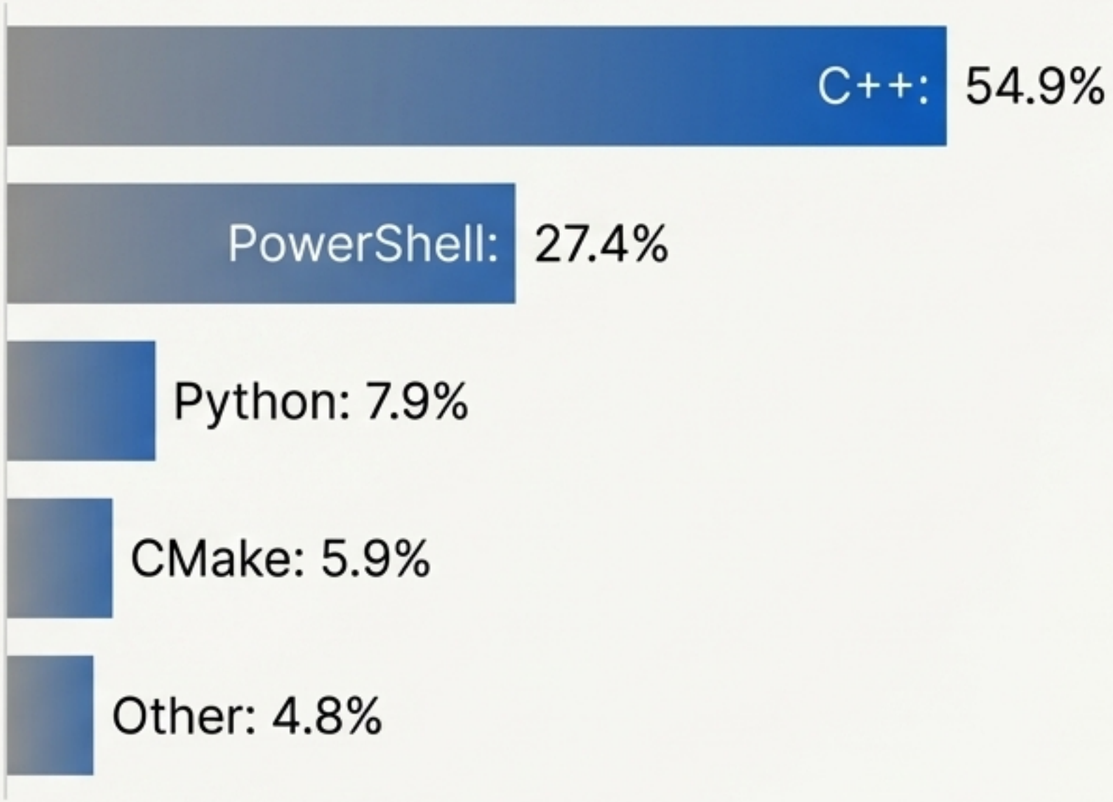
`C++` `DirectX 11/12` `Vulkan` `MinHook`

`FSR1` `Waifu2x` `ncnn` `CMake`

License

- ***Type****: MIT License

Language Distribution



Making games look better, run faster.