POLITECNICO DI MILANO STMICROELECTRONICS A.Y. 2025 / 2026

MULTIDISCIPLINARY PROJECT

Documenting neural-network deployment

A report on how a generated Doxygen site became a designed, reproducible documentation architecture.

HARDWARE
STM32N6570-DK

SUBJECT
The architecture of the documentation site

AUTHORS
Patrizio Acquadro, Giacomo Colosio, Sebastiano Colosio, Tito Nicola Drugman

SUPERVISORS
Prof. Cristina Silvano, Dr. Marco Ronzani

23 annotated source files 3 model architectures 110+ generated HTML pages

Executive Abstract

This report describes the documentation architecture created for the STM32N6 neural-network deployment project. The technical work behind the project was not only to run models on the STM32N6570-DK Discovery Kit; it was also to make that work understandable, reproducible and maintainable for readers who did not share the team's tacit context.

The resulting site is a self-contained documentation artifact. It combines a six-chapter narrative, an execution-order code reference and an architectural module view over the same source tree. The central design decision is simple: different readers ask different questions. A supervisor asks why the result matters; a developer asks where the implementation lives; a future maintainer asks which files belong to the host pipeline and which belong to the target firmware. The documentation gives each reader a dedicated entrance without splitting the project into separate sources of truth.

Code that runs is half the work. The other half is making it readable.

PAGES	SOURCE FILES	READING MODES
110+	**23**	**3**
generated HTML pages	13 Python, 10 C/H	narrative, code, layers

Contents

Executive Abstract 1

1 Project Context: Real-Time AI Without a Server 2

2 Three Architectures, One NPU 2

3 The Documentation Problem 4

4 Three Reading Modes 4

5 Part 1: Narrative as a Spine 5

6 The Five-Beat Chapter Rhythm 6

7 Part 2: Code Reference as an Execution Timeline 7

8 Part 3: Module Groups as Architecture 8

9 Cross-Linking: Every Page Leaves Three Doors Open 10

10 Build Pipeline: Generated, Then Curated 10

11 Visual System: Meaning Before Decoration 11

12 Outcome: Three Audiences, One Source of Truth 11

13 Acceptance Criteria for the Documentation Artifact 13

14 Conclusion 13

Source Base 13

1 Project Context: Real-Time AI Without a Server

HARDWARE AND GOAL

The project studies neural-network deployment on the STM32N6570-DK, a microcontroller development board built around the STM32N657X0H3Q device. The board combines a Cortex-M55 CPU, the Neural-ART accelerator, local SRAM, external Flash, a camera path and an LCD output path. In practice, that means the deployed application must respect embedded constraints while still delivering inference fast enough for interactive vision workloads.

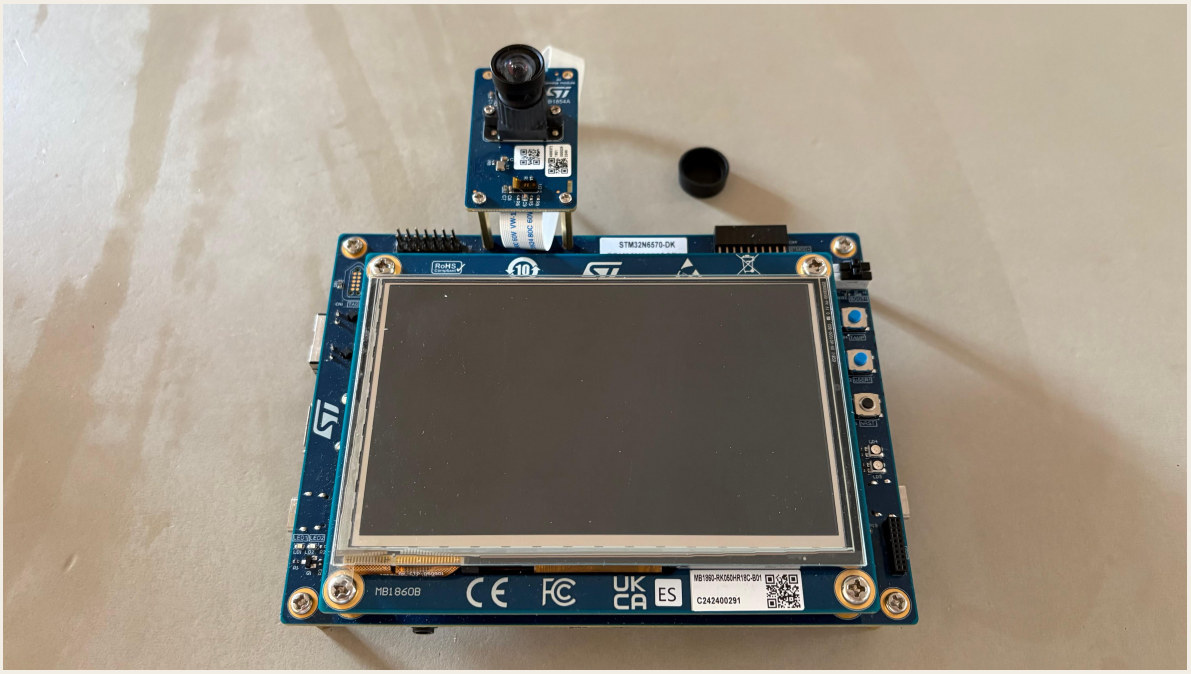

Figure 1: STM32N6570-DK board used as the deployment target.

Table 1: Hardware facts used throughout the documentation.

Component	Role in the project
Cortex-M55	General-purpose control, firmware orchestration, fallback execution and peripheral management.
Neural-ART NPU	600 GOPS INT8 accelerator used for neural-network inference on supported operations.
Memory	4.2 MB SRAM plus 128 MB external Flash, shaping model placement and runtime buffers.
I/O	MIPI CSI-2 camera input and LCD output used for real-time pose-estimation demos.

The documentation does not treat these facts as isolated specifications. It uses them as explanatory anchors. A reader first learns what edge deployment requires, then sees why the board matters, then follows how the host pipeline turns a model into firmware artifacts for the target.

2 Three Architectures, One NPU

EXPERIMENTAL FRAME

The project deployed three different model families to test how a CNN-centric embedded accelerator behaves

under different operator mixes. The result is not just a performance table; it is the technical reason the documentation needs a narrative layer. Without explanation, the numbers can be misleading. TinyBERT, for example, is smaller than MoveNet in weight size, yet it performs worse on the NPU because its operation pattern is less aligned with the accelerator.

Table 2: Model outcomes summarized in the documentation and presentation.

Model	Architecture	NPU offload	Latency
MoveNet Lightning	CNN, MobileNetV2 style backbone for single-person pose estimation	94.7%	22 ms
YOLOv8n-pose	CNN backbone with detection head for multi-person pose estimation	87.9%	32 ms
TinyBERT	Transformer model used to probe attention-heavy operator behavior	64.4%	>100 ms

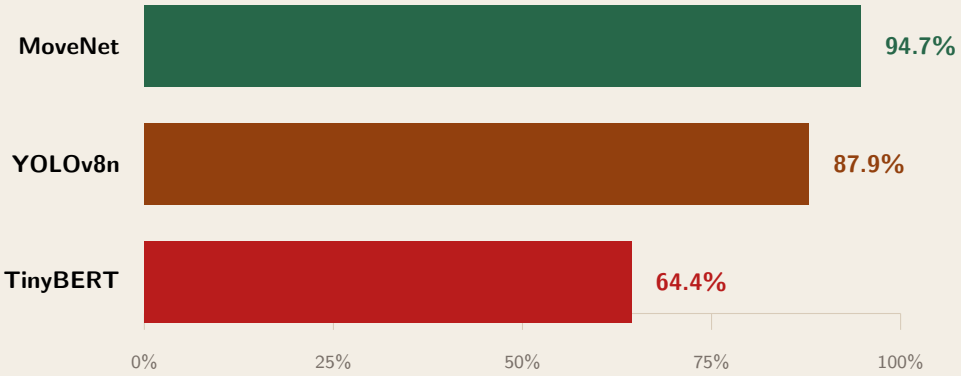

Figure 2: The documentation turns raw offload percentages into an architectural explanation.

This distinction is important. A simple benchmark report could stop at the table. A documentation site must make the table traceable: from model, to toolchain, to generated report, to source file, to function-level graph.

3 The Documentation Problem

FROM WORKING TO READABLE

The problem was not the absence of information. It was the fragmentation of information. The project relied on ST application notes, Edge AI tooling, ModelZoo material, CubeIDE project generation, Python preprocessing scripts, C firmware and target-specific memory constraints. Each source had value, but none alone explained the full path from model selection to board execution.

Table 3: Documentation risks and the design response.

Risk	Failure mode	Documentation response
Heterogeneous sources	A future reader must reconstruct context across vendor docs, scripts, generated code and team notes.	A single documentation site establishes the project sequence and links back into concrete source pages.
Tooling churn	Tool versions and command details can drift, making old deployment recipes hard to replay.	Toolchain and workflow chapters record the exact path and expected artifacts.
Tacit knowledge	Choices such as configuration flags, buffer layout and deployment order stay in team memory.	Narrative chapters explain why a step exists before linking to the implementation.
Host/target split	Python code and C firmware feel like two separate projects rather than one deployment system.	Part 2 orders files by execution; Part 3 groups files by architectural layer.

The documentation thesis is therefore architectural: readability is produced by structure, not volume. The site contains more than 110 generated pages, but the design goal is to prevent the reader from experiencing it as a pile of pages. Every major page should answer one of three questions: why does this step exist, when is this file executed, or which subsystem owns this responsibility?

4 Three Reading Modes

ONE SOURCE TREE, THREE DOORS

The site is organized around three complementary reading modes. They are not three separate documents; they are three views over the same project. This is the central architectural move of the documentation.

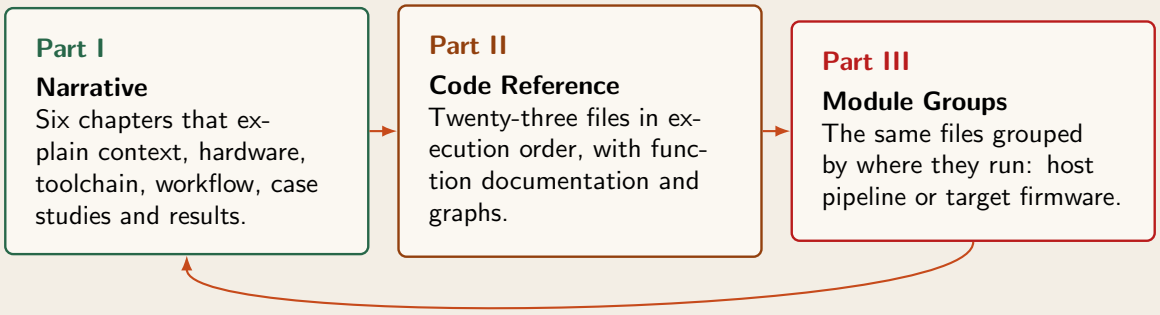

Figure 3: The three-part structure makes the site navigable by question rather than by file count.

Table 4: Reader contracts for the three documentation modes.

Mode	Primary reader	Question answered
Part 1: Narrative	Supervisor, reviewer, first-time reader	Why does the project matter and what must be understood first?
Part 2: Code Reference	Future developer	Where is the implementation and what calls what?
Part 3: Module Groups	Maintainer or subsystem owner	Which files belong to host automation and which run on the STM32 target?

5 Part 1: Narrative as a Spine

SIX CHAPTERS IN ORDER

Part 1 is the documentation spine. It starts from concepts, moves through hardware, introduces the toolchain, describes the deployment workflow, compares case studies and closes with results. This order matters because each chapter assumes only the context introduced earlier.

Table 5: Narrative chapter sequence.

Chapter	Purpose
Introduction	Establish edge computing, embedded systems, neural networks and INT8 quantization.
Hardware	Explain the STM32N6570-DK, Neural-ART NPU, memory hierarchy and camera path.
Toolchain	Connect ST Model Zoo, ModelZoo Services, ST Edge AI Core and STM32CubeIDE.
Workflow	Show the deployment sequence from configuration to flashing and execution.
Case Studies	Compare MoveNet, YOLOv8n-pose and TinyBERT with measured behavior.
Results & Analysis	Interpret offload, latency, memory and architectural limits.



Figure 4: Part 1 provides a textbook-style entry point before the reader enters source-level detail.

This is where the report style mirrors the presentation most directly. The presentation frames Part 1 as a first-year-friendly path through a complex project. The report keeps that principle: a reader should not have to know what quantization, DCMIPP, or NPU offload mean before opening the documentation. Those ideas are introduced before they become measurements.

6 The Five-Beat Chapter Rhythm

PATTERN AS USABILITY

A chapter is not only a container of text. In this documentation, it follows a repeatable rhythm that lowers cognitive load. The reader learns what kind of information comes next and can scan the site without relearning the page grammar each time.



Figure 5: The five-beat pattern turns chapter design into a repeatable reading interface.

1. **Hero header.** A dark banner gives the chapter number, title and promise.
2. **Concept in plain language.** Definitions and intuition arrive before acronyms and numbers.
3. **Diagram.** Inline visuals explain systems such as memory hierarchy and NPU execution.

4. **Concrete numbers.** Measured results are introduced with their source context.
5. **Bridge to code.** The chapter closes by pointing to the file or function that implements the concept.

7 Part 2: Code Reference as an Execution Timeline

NOT A FILE LISTING

Generated code documentation often defaults to alphabetical file lists. That is useful for lookup but weak for learning. Part 2 changes the organizing principle: files appear in the order the program actually calls them, moving from host-side Python automation to target-side C firmware.

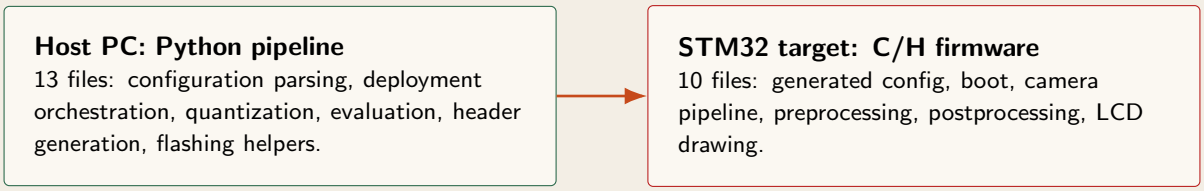

Figure 6: Part 2 presents the source in execution order, from host preparation to target runtime.

Table 6: Execution timeline captured by Part 2.

Stage	Representative responsibility
Configuration	Read and validate `user_config.yaml`; return a global deployment configuration.
Deployment orchestration	Choose operation mode, invoke ST Edge AI Core, generate headers, build, sign and flash.
Quantization and evaluation	Convert models to INT8 and validate accuracy with model-specific metrics.
Firmware setup	Initialize clocks, memory, NPU runtime, camera input and main loop.
Real-time inference	Capture frames, preprocess, run inference, decode outputs and draw results on LCD.

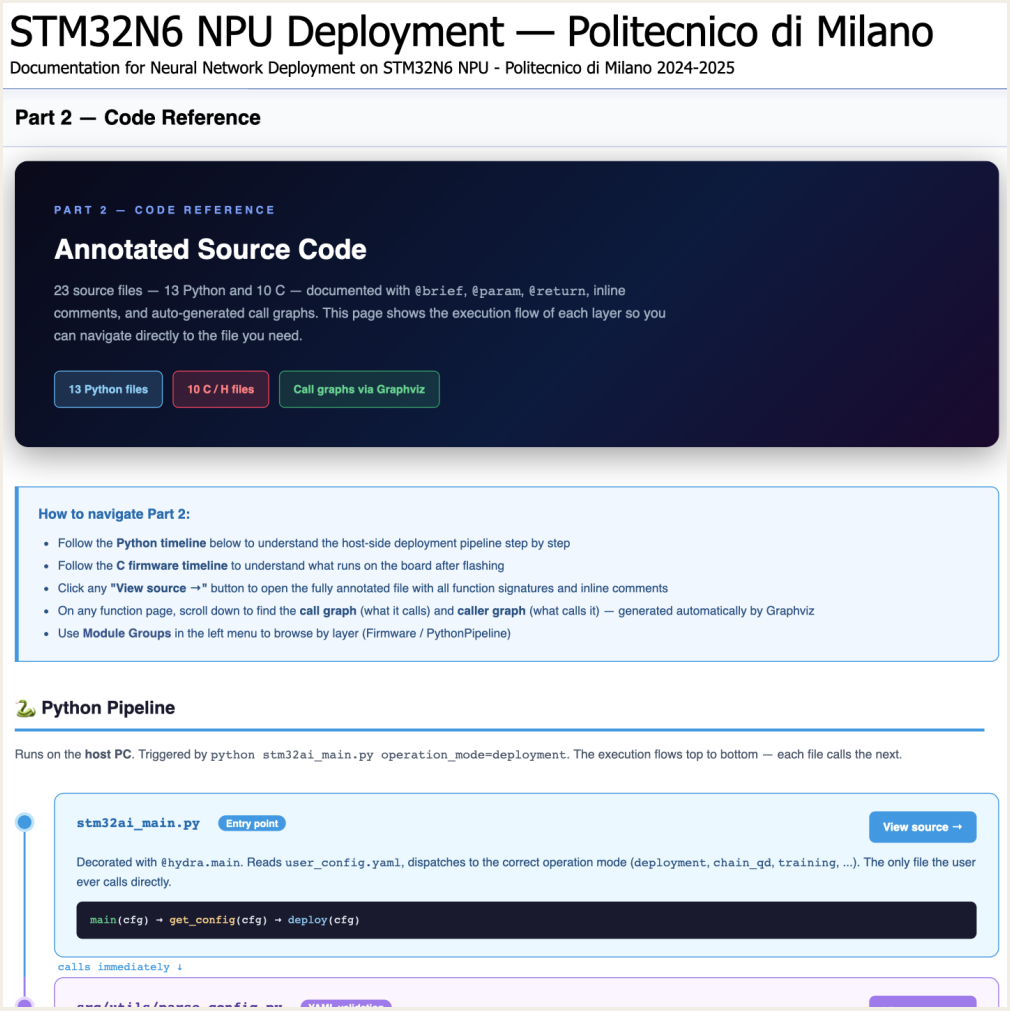

Figure 7: The code reference exposes files, functions and generated call/caller graphs.

The reference layer is deliberately close to the code. Doxygen annotations document functions with brief descriptions, parameters and return values. The Graphviz output adds call and caller graphs below source pages, so the reader can move forward and backward through execution without manual diagrams that would drift.

8 Part 3: Module Groups as Architecture

SAME CODE, DIFFERENT QUESTION

Part 3 reorganizes the same source material by responsibility. This is the maintainer view. Instead of asking when a file executes, it asks where the file belongs architecturally.

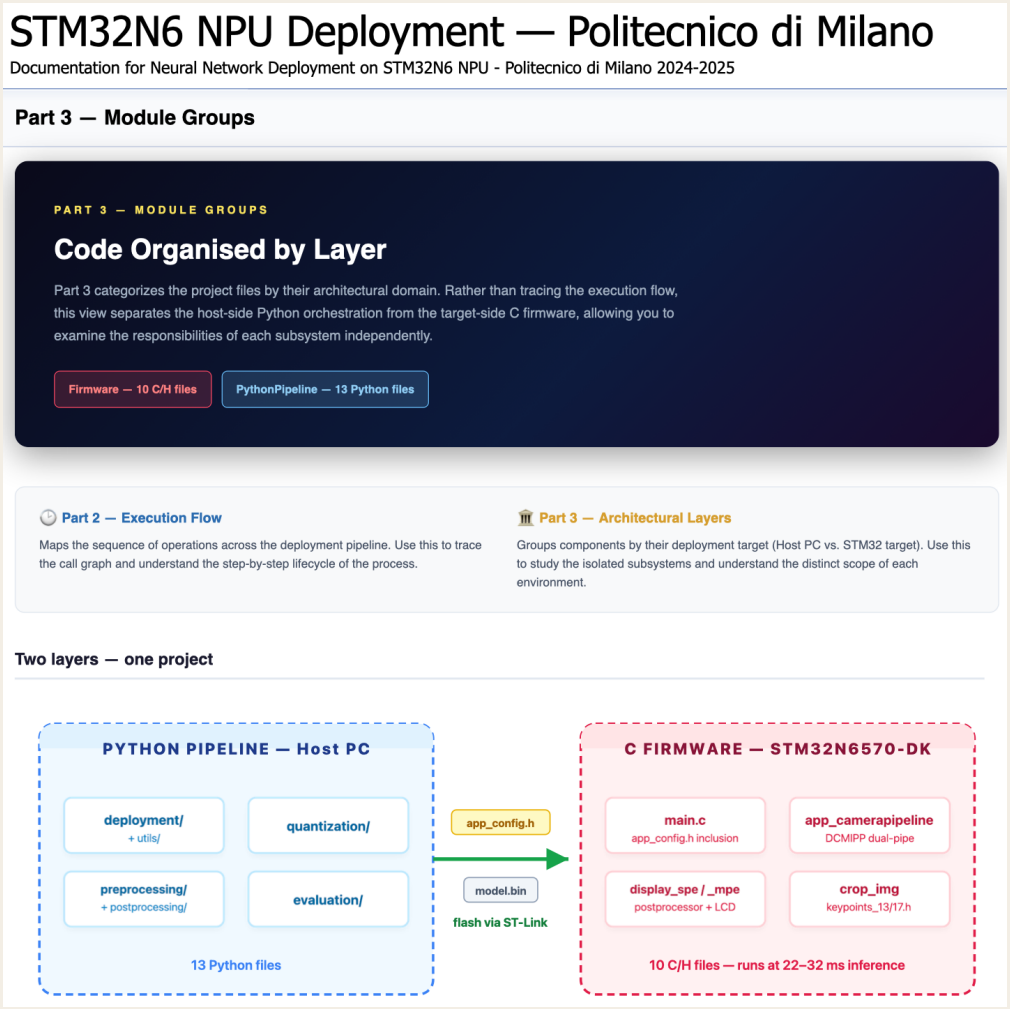

Figure 8: Part 3 separates the host pipeline from the firmware application.

Table 7: Architectural grouping of documented files.

Group	Environment	Documentation value
Python Pipeline	Host PC	Collects data parsing, model management, quantization, evaluation, code generation and external tool invocation.
Firmware	STM32N6570-DK	Collects target runtime files for camera acquisition, NPU execution, postprocessing and display output.

This split matters because the project is a bridge between two execution domains. The documentation must preserve that bridge without flattening it into a single list. A student extending the deployment script and a student changing the camera pipeline should not have to scan the same unrelated files to find their starting point.

9 Cross-Linking: Every Page Leaves Three Doors Open

NAVIGATION MESH

The site avoids dead ends. The navigation design is a mesh: concepts link to files, files link to functions, functions link to graphs and grouped pages link back to their domain. The same entity can be reached from above, below, or across.

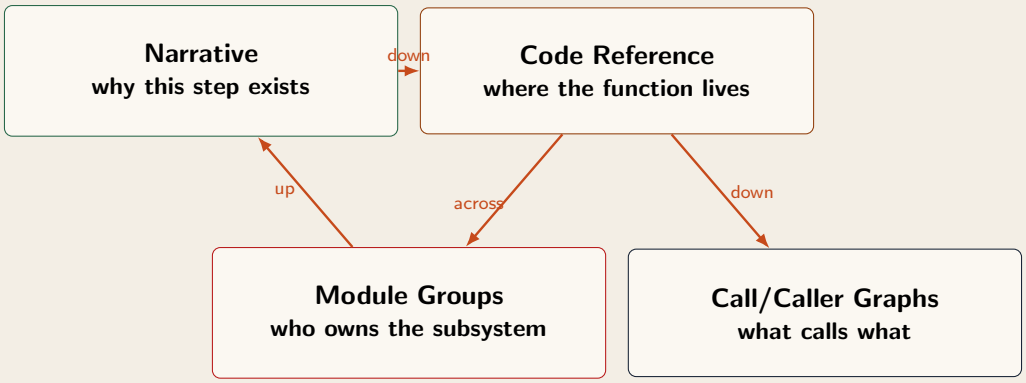

Figure 9: The documentation mesh supports conceptual, source-level and architectural navigation.

Table 8: Navigation directions and their meaning.

Direction	Reader action
Up	Return from a source file to the chapter that explains the concept.
Down	Move from prose to a file, then from a file to a function or graph.
Across	Jump from execution order to host/target ownership.

10 Build Pipeline: Generated, Then Curated

REPRODUCIBILITY

The documentation is generated rather than hand-written page by page and the source annotations, Doxygen pages, Graphviz graphs and post-processed static site are all part of one build story.

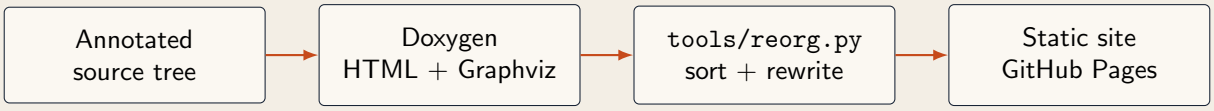

Figure 10: Build pipeline used to make the generated output readable and publishable.

Table 9: What the post-processing stage contributes.

Action	Why it matters
Move assets into folders	Doxygen's flat output becomes a browsable repository with `css/`, `js/`, `images/`, `graphs/` and `search/`.
Rewrite references	HTML, CSS, JavaScript and SVG links continue to work after the move.
Keep search intact	The Doxygen search folder remains compatible with the generated site.
Make the process idempotent	Maintainers can rerun the script after regeneration without manually repairing paths.

The build pipeline is also an argument about maintenance. Manual screenshots or hand-maintained diagrams would make the site attractive but fragile. Generated graphs and a repeatable reorganization step make the documentation attractive without disconnecting it from the code.

11 Visual System: Meaning Before Decoration

EDITORIAL DESIGN

The presentation uses a strong editorial vocabulary: cream backgrounds, dark contrast, copper section markers, generous serif headings and narrow structural rules. This report follows that language, but only where it supports the documentation thesis. The visual system is not ornamental; it is another way of making the information hierarchy readable.

Table 10: Visual conventions translated from the presentation into the report.

Convention	Use in the report
Cream and dark contrast	Separates title/quote pages from analytical sections while preserving a coherent tone.
Copper accents	Marks section openers, arrows and the documentary thread through diagrams.
Large serif headings	Signals editorial importance without turning every page into a slide.
Slim rules	Create structure without heavy boxes or decorative clutter.
Vector diagrams	Keep architecture, build flow and cross-linking sharp in print.

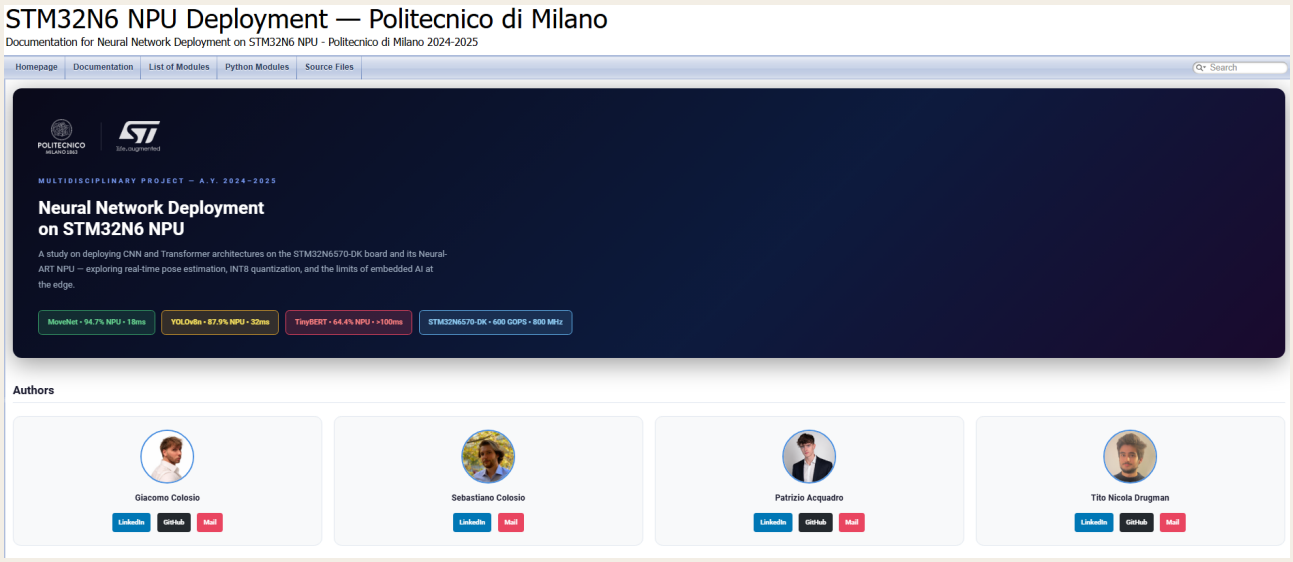

Figure 11: The live site homepage is the user-facing entry point for all three documentation modes.

12 Outcome: Three Audiences, One Source of Truth

WHAT THE DOCUMENTATION ACHIEVES

The documentation succeeds when different audiences can enter through different doors and still land on the same source of truth.

Table 11: Audience outcomes.

Audience	Entry point	Successful outcome
Supervisor or reviewer	Part 1 narrative	Understands the full project arc without opening a source file.
Future student or developer	Part 2 code reference	Traces a workflow step to the exact file, function and call graph.
Subsystem owner	Part 3 module groups	Sees the host and target boundaries without reading unrelated files.

REPRODUCIBLE
Same source
same generated output

VERSIONED
Git history
docs travel with code

SEARCHABLE
Full text
110+ pages indexed

This is why the documentation is best understood as an architecture. It has interfaces, invariants and users. Its core invariant is that explanation and implementation must stay connected: a concept should lead to a file, a file should lead to functions and functions should expose the graph around them.

13 Acceptance Criteria for the Documentation Artifact

HOW TO JUDGE QUALITY

The report also makes explicit how this kind of documentation should be judged. The question is not whether the site is large. The question is whether it reduces the time needed to understand, reproduce and extend the project.

Table 12: Quality criteria for the documentation site.

Criterion	Signal	Failure if absent
Completeness	All 23 source files are annotated and reachable from the site.	The documentation becomes a partial tour rather than a source reference.
Traceability	Concepts link to files and files expose call/caller graphs.	Readers must guess where prose becomes implementation.
Reproducibility	Doxygen and `tools/reorg.py` rebuild the site from source.	The site can drift from the code and lose trust.
Audience fit	Narrative, code and layer views answer different reader questions.	One audience is served at the expense of another.
Maintainability	Generated output is organized into stable folders and validated links.	Static assets break or become hard to review in Git.

14 Conclusion

DOCUMENTATION AS ENGINEERING

The STM32N6 project could have been documented as a long chronological report: choose a model, quantize it, build firmware, flash the board, record results. That would have preserved the sequence but not the architecture. The better choice was to design a documentation system with three durable reading modes.

The narrative mode explains why the work matters. The code-reference mode shows where the implementation lives. The module-group mode makes ownership visible. Together, they turn a complex embedded AI workflow into a navigable artifact.

> *Three parts beat one long document when the project has more than one kind of reader.*

The main lesson is transferable: documentation for technical projects should be treated as an interface. It needs structure, tested links, repeatable builds, and a clear contract with its users. In this project, that contract is now visible: 23 source files, three model architectures, one NPU deployment workflow and a documentation site that makes the whole system readable.

Source Base

This report was written from the final presentation and the local documentation repository. The main local evidence sources are `README.md`, `index.html`, `part1_page.html`, `part2_page.html`, `part3_page.html`, `workflow_page.html`, `casestudies_page.html`, `results_page.html` and `tools/reorg.py`. The live documentation URL is: PatrizioAcquadro.github.io/stm32n6-docs/.