OpenCV

Discrepancy Analysis

eLand

Eric Rodrigues, Laura Marin, Alp Baran Sirek, Negar Khalilazar, Danny Le

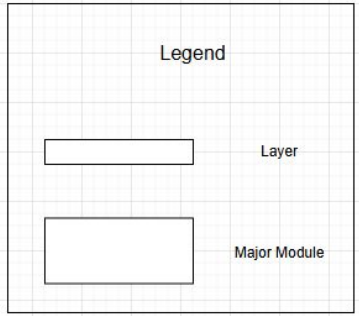
Introduction

1. **Objective:** Analyze how the conceptual architecture from A1 aligns with the concrete architecture extracted in A2
2. **Subsystem Under Study:** G-API, OpenCV's graph-based computation engine
3. **Approach:** Reflexion analysis (comparing expected relationships to the actual implementation)
4. **Outcome:** Identify discrepancies, explain their rationales and propose refinements to both models

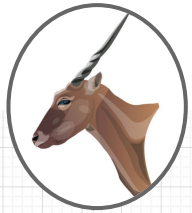
Revised Breakdown
&
Reflexion Analysis

Reflexion Analysis

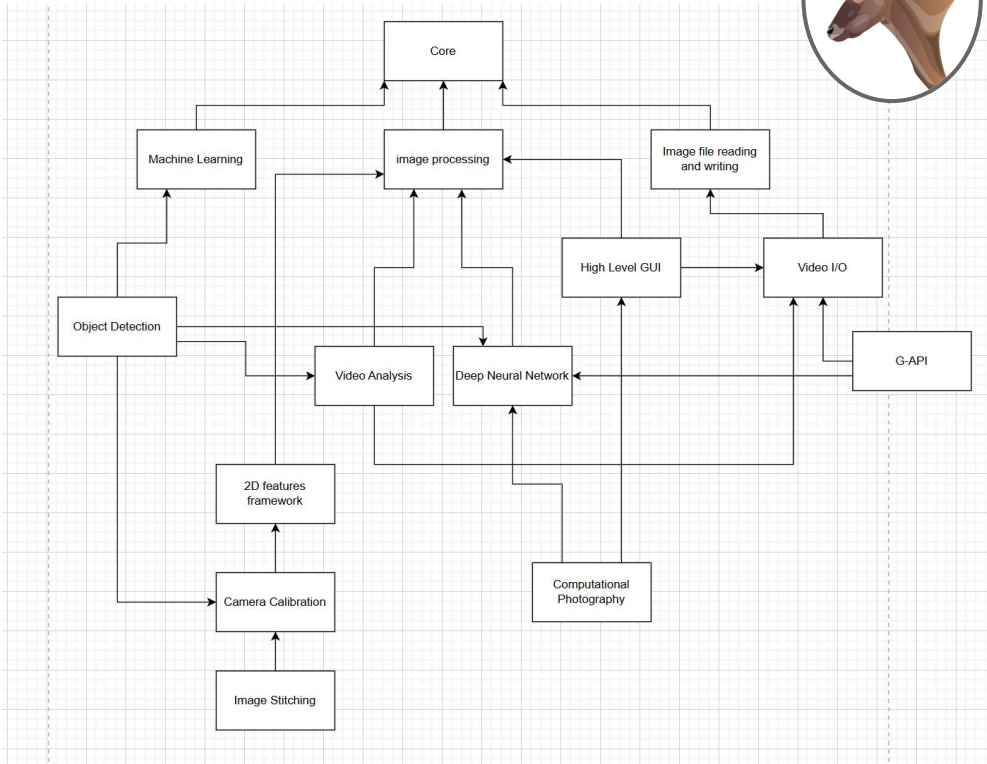
Layer 4: Advanced Functionality				
Object Detection	Deep Neural Network	Machine Learning	Image Stitching	G-API
Layer 3: Analysis and Feature Extraction				
Video Analysis	2D Features Framework	3D Reconstruction		
Layer 2: Base Processing and I/O				
Image Processing	Video I/O	High level GUI	Computational Photography	
Layer 1: Core				
Core Functionality				

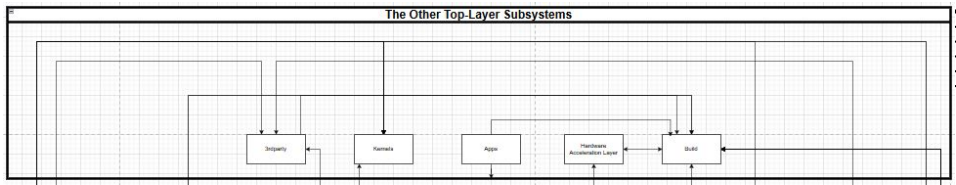


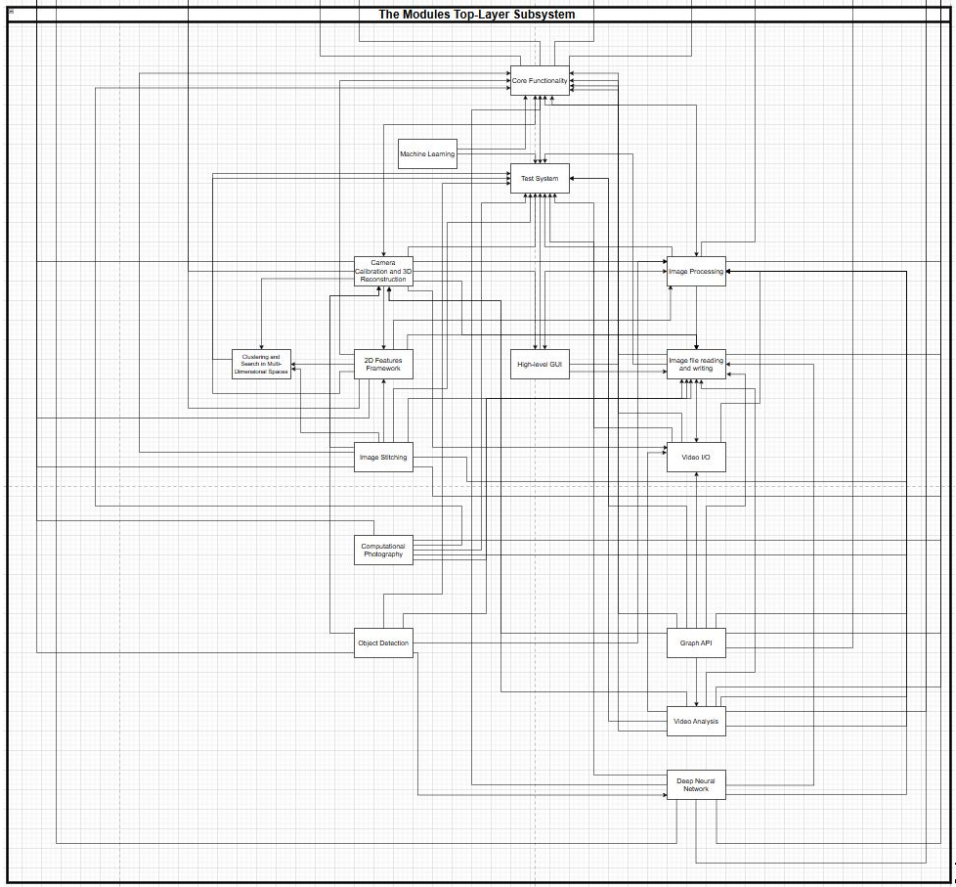
Top Level: Conceptual



- Modular system, designed for scalability.
- Uses a layered architecture design pattern.



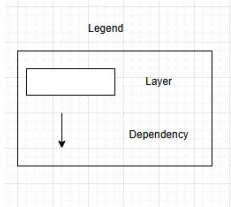




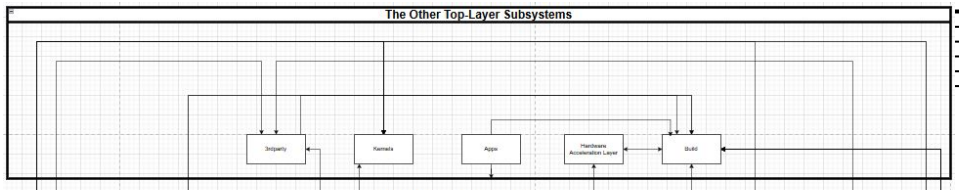
?

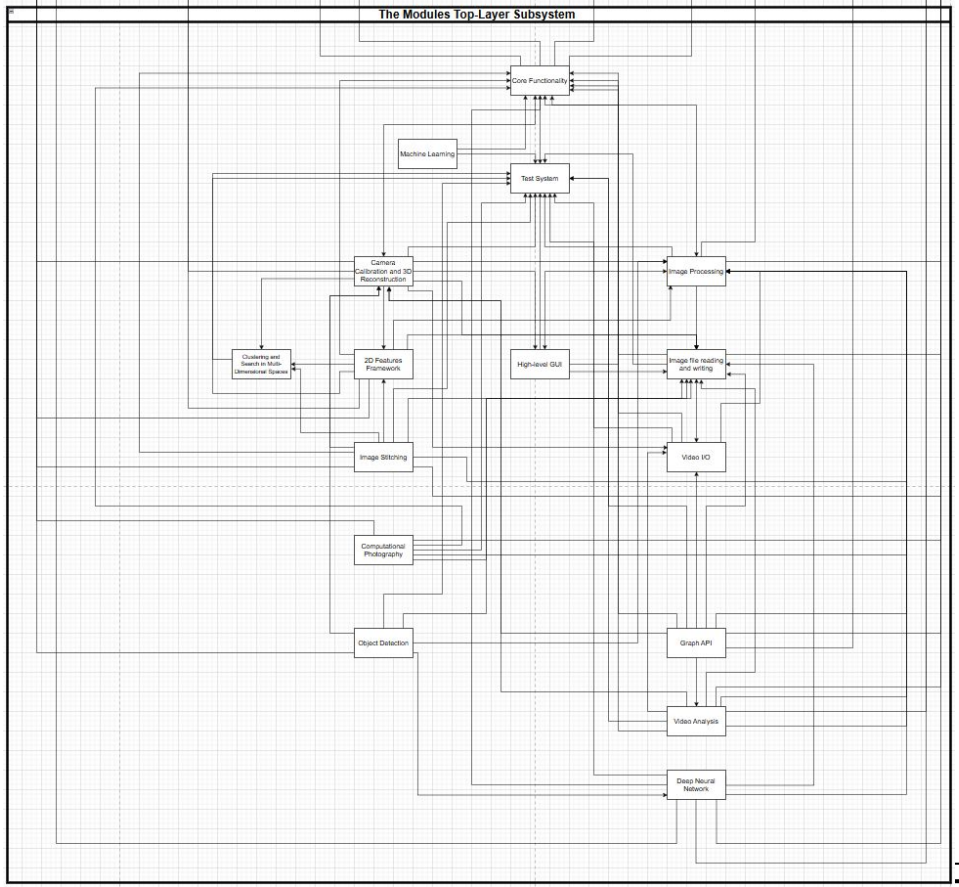
Top Level: Concrete

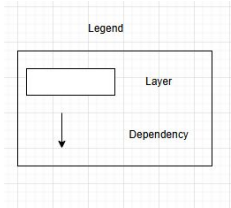
- More complex than conceptual architecture depicted.
- More interdependencies between modules; and is not as hierarchical.
- Dependencies to other top-level subsystems, not found in the conceptual architecture.



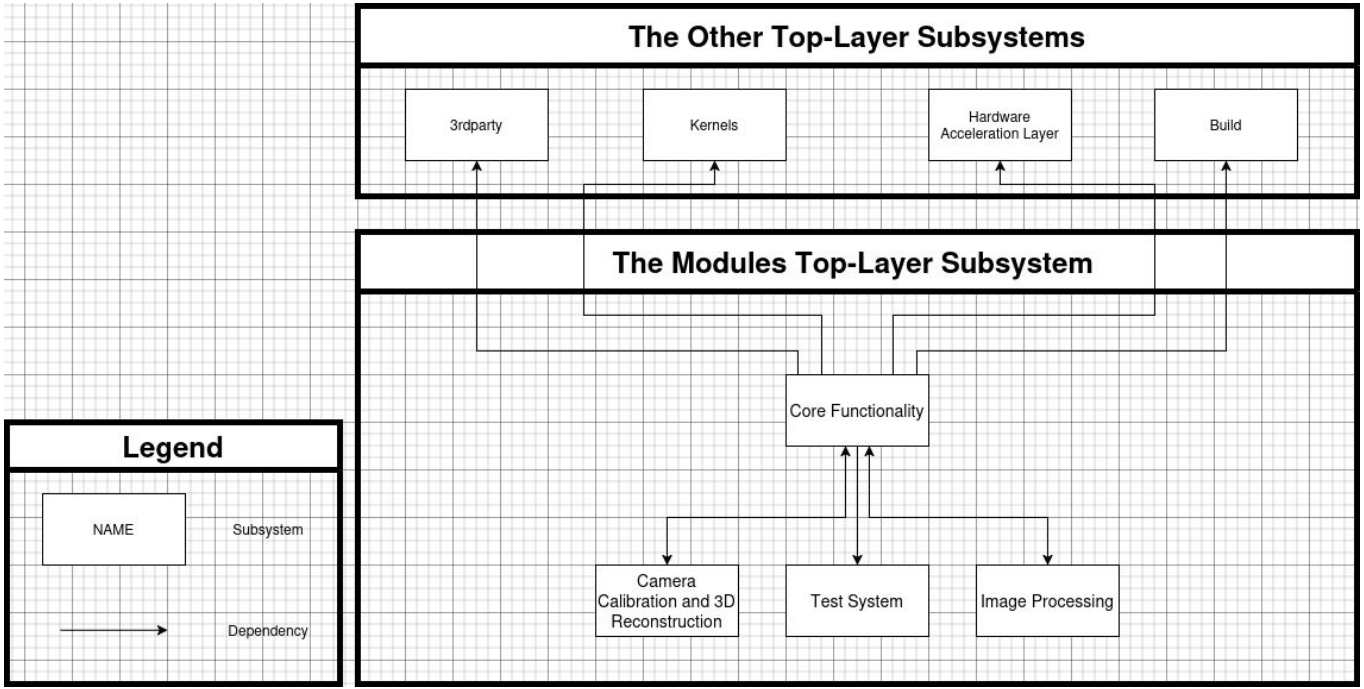
Reflexion Analysis







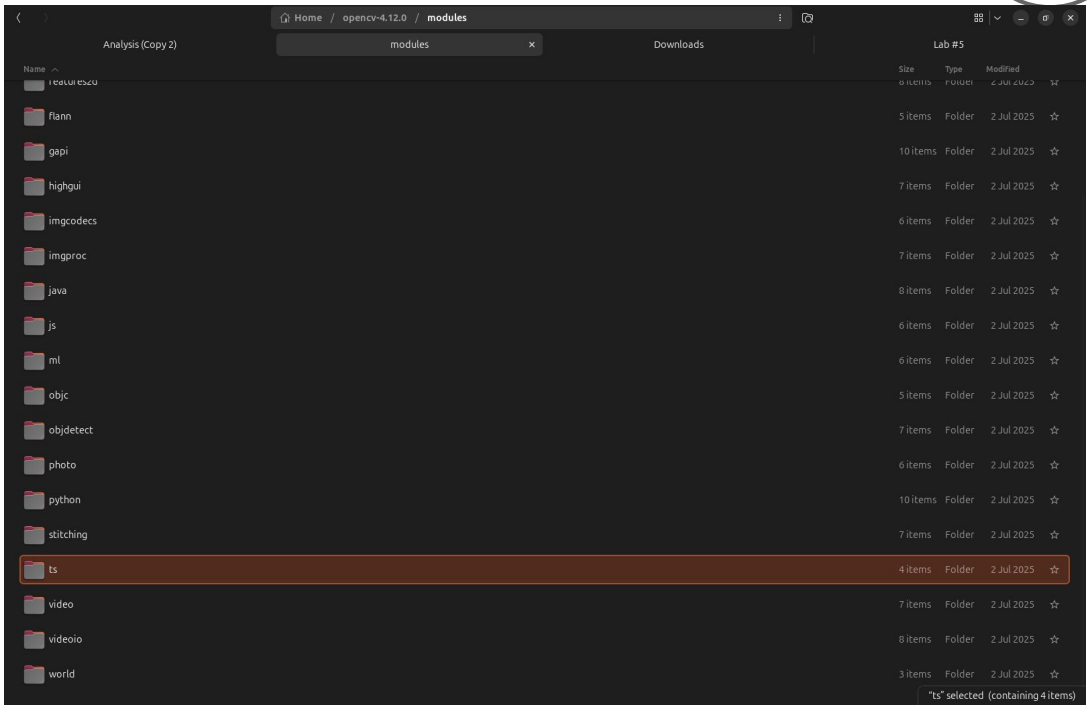
Discrepancies of Top Level



Top Level: Concrete

- Undocumented “ts” (test support) module.

- Classes for tests:
  - “TS” class for managing tests.
  - “BaseTest” class contains methods for testing.
  - “ArrayTest” for testing dense arrays.
  - “BadArgTest” for testing bad argument handling.



Discrepancies of Top Level

Which?	G-API module compile error with GCC 11 in gapi_async_test.cpp (GitHub #19678)
Who?	Aleksey Churbanov (OpenCV Core)
When?	March 2021
Why?	G-API's test setup broke with new GCC standards; Aleksey fixed CMake and code to resolve this. Shows how toolchain updates expose hidden dependency issues, and how concrete code ends up diverging from original modular intentions.

Top-Level: Rationale

- Why we saw more edges than the layered picture suggested
  1) shared utilities in core
  2) top layer subsystems that many modules touch
  3) test system (ts)

- How we validated it
  1) LSEdit graphs & Understand traces
  2) CMakeLists inspection
  3) checks against commit history for dependency changes

G-API: Conceptual

- Conceptual Architecture
- Expected

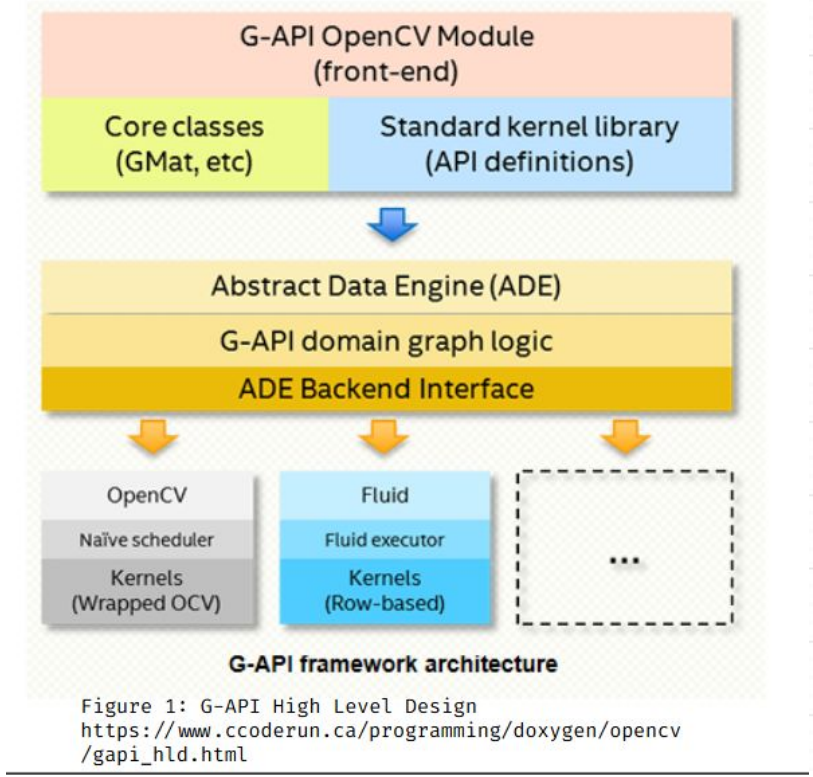
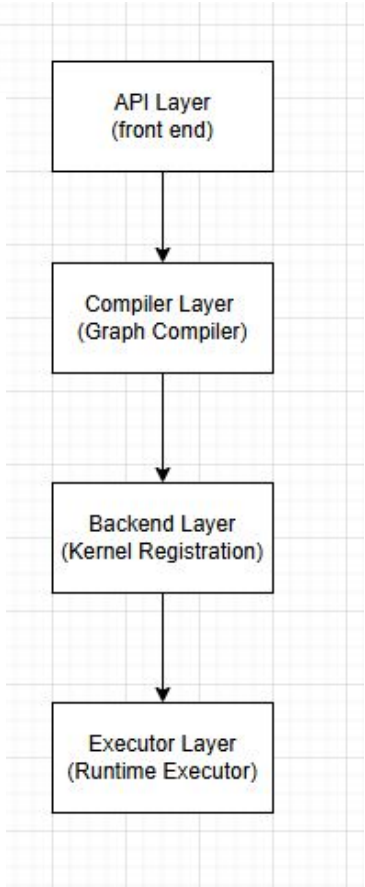
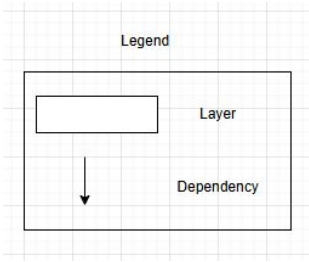


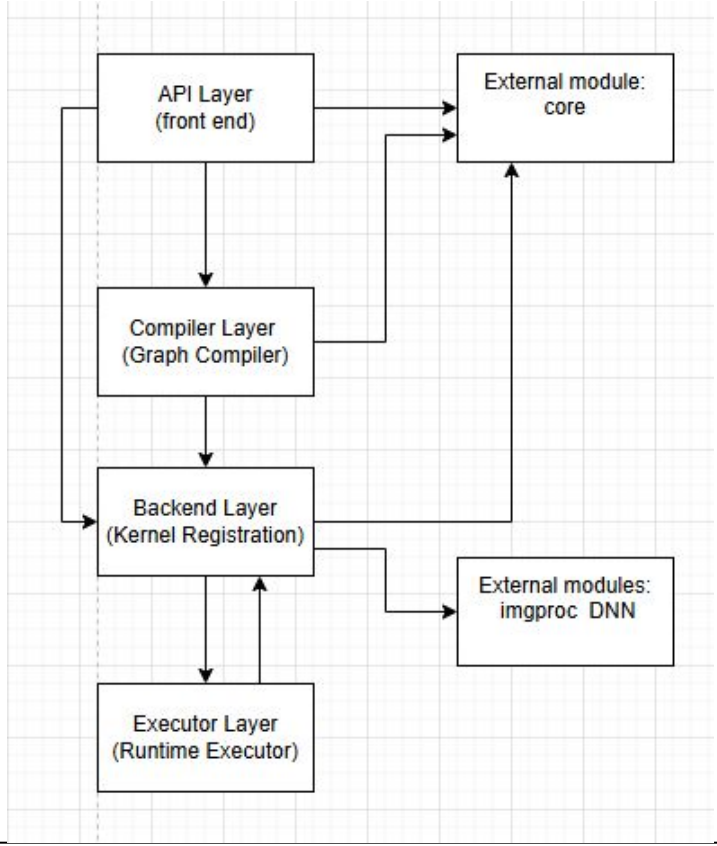
Figure 1: G-API High Level Design
https://www.ccoderun.ca/programming/doxygen/opencv/gapi_hld.html

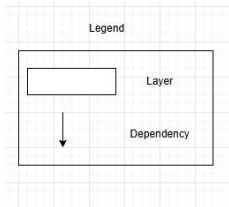




G-API: Concrete

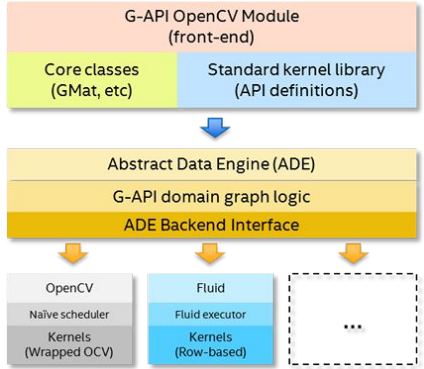
- Implementation and Mapping
- Actual

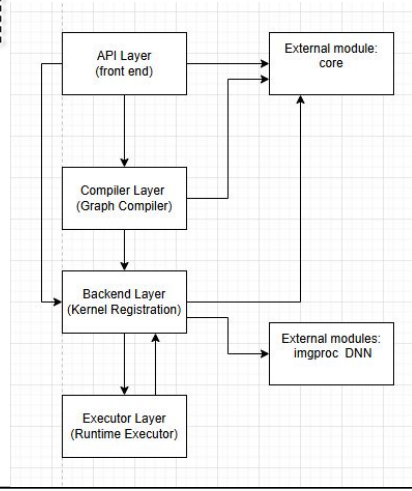




Discrepancies of G-API

- The major difference between our two models was internal complexity
- In our conceptual diagram graph compilation looked like a single step
- The concrete system however performs multiple phases





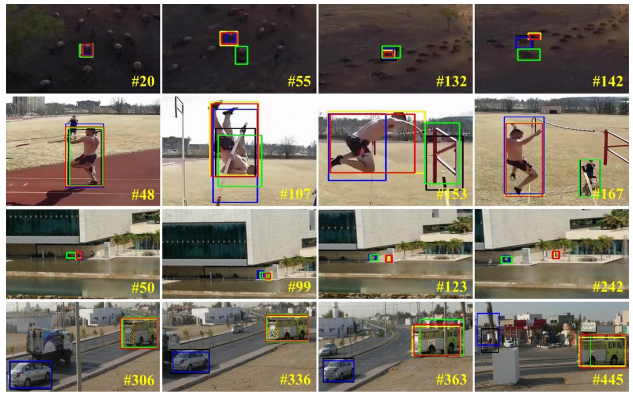
Discrepancies of G-API

Which?	The executor layer depends on internal compiler metadata, creating an unexpected reverse dependency. Instead of the conceptual one-direction flow.
Who?	Dmitry Matveev Alexander Smorkalov
When?	November 2018
Why?	The dependency exists because reading compiler internals makes G-API significantly faster and more efficient.

G-API Comparison

- Conceptually we described G-API as a high-level, graph-based layer where developers define image-processing pipelines declaratively

- The concrete architecture we visioned matched the overall structure but included much more detail



Summing Up

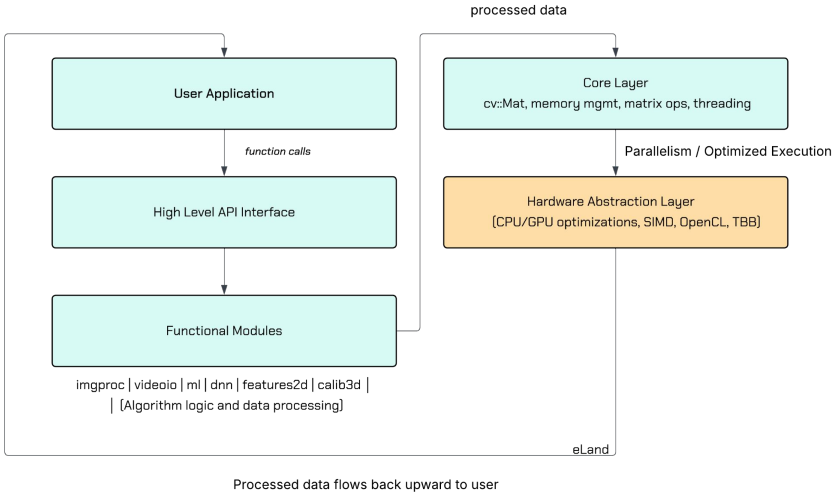
- Our conceptual architecture should be refined to show the more complex layers
- On the concrete side documentation and backend organization could be improved
- Overall the concrete architecture does validate our conceptual understanding

G-API: Rationale

- What matched
  1) clear layering in code and directories
  2) strong dependency on core as expected

- Where differences came from
  1) compilation is multi-phase
  2) backend registration and stateful kernels introduce optional ties

Use Cases and Conceptual Control Flow



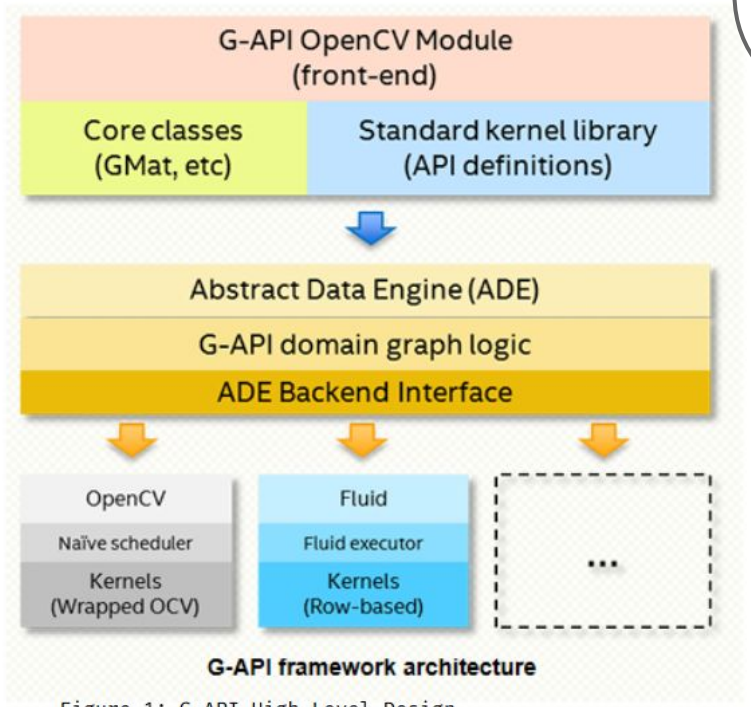
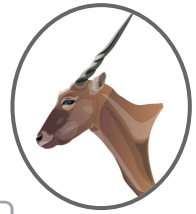
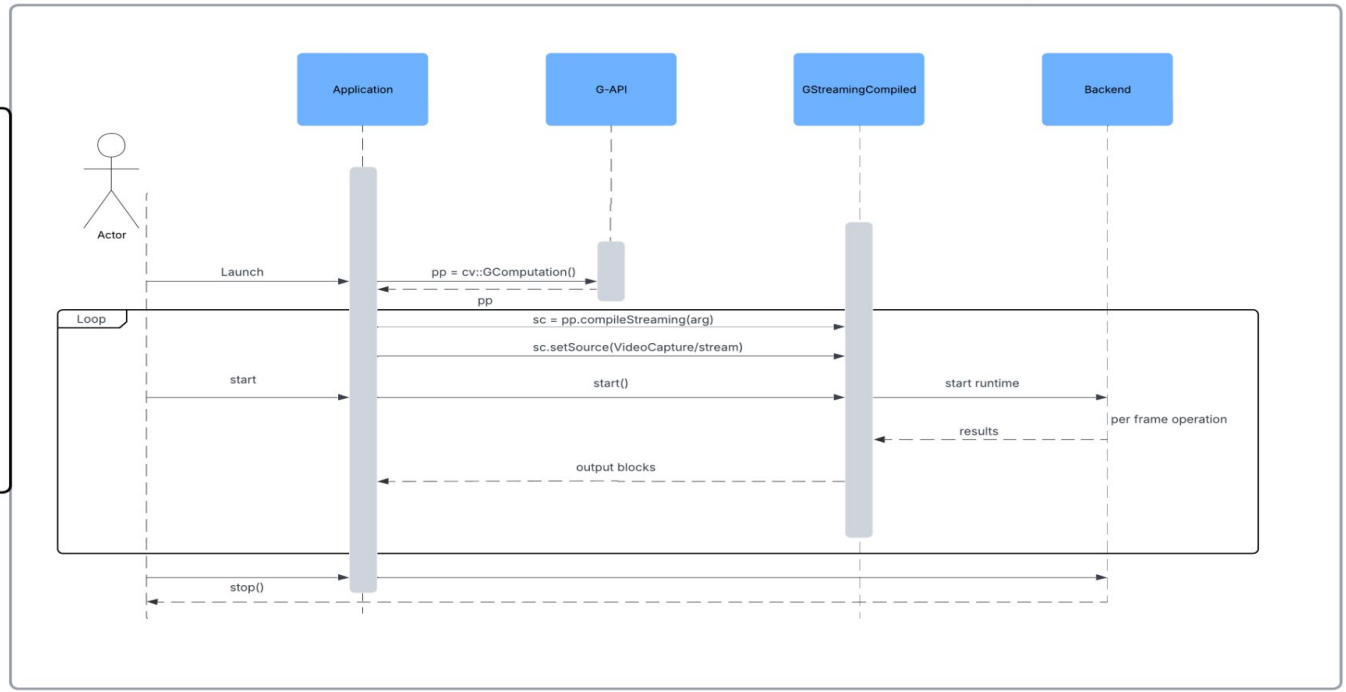


Figure 1: G-API High Level Design
https://www.ccoderun.ca/programming/doxygen/opencv/gapi_hld.html

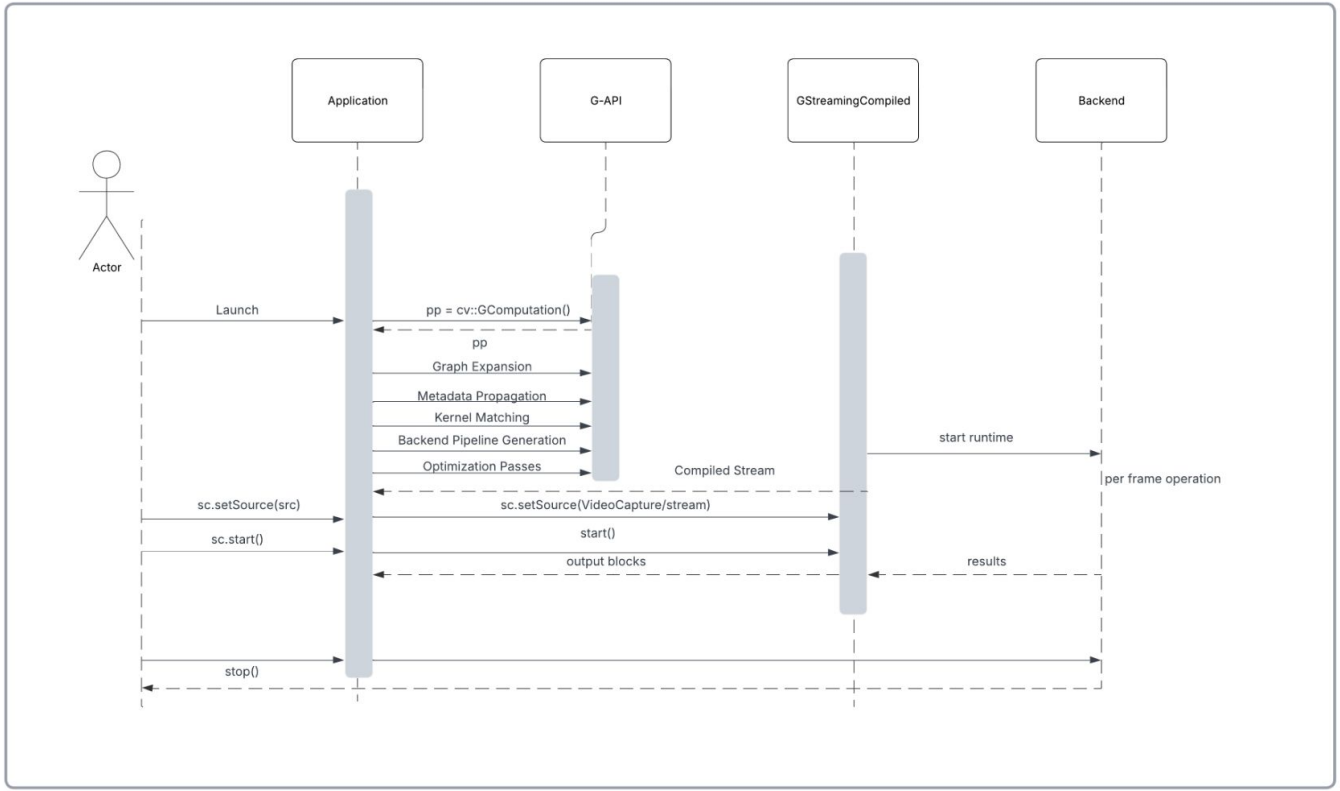
Use Cases



G-API Webcam Object Detection with Overlay (Streaming)



Refined A3 Understanding (Based on Concrete Architecture)



legend

return/ack/data

call/request

Effects of Team issues and Concurrency & Limits of Our Findings

Conclusion

Lessons Learned

1. Conceptual architectures are intentionally simplified views
2. Modularity in theory can be very different in practice
3. The importance of abstraction layers
4. The value of architectural tooling

Questions?