Using GitHub Copilot with VSCode to Build Raspberry Pi 5 RTSP YOLO Pipeline

Step-by-Step Guide: Building a Modular Python RTSP Detection Pipeline for Raspberry Pi 5 with VSCode & GitHub Copilot

---

Introduction

Building an advanced, modular Python project for the Raspberry Pi 5 (8GB) with a Hailo8L AI Hat-such as the pi-live-detect-rtsp system-requires a robust workflow that leverages modern coding tools. Visual Studio Code (VSCode) and GitHub Copilot, with their AI-driven features, streamline every phase of development, from repository scaffolding and remote coding to documentation, service orchestration, and CI/CD. This guide will deliver a comprehensive, proven workflow based on the provided project outline. We will detail each phase, referencing relevant sources, practical recommendations, and Copilot-specific strategies, ensuring you can maximize development speed while maintaining high code quality and reliability for a multi-service, edge-AI streaming solution.

---

1. Repository Scaffolding in VSCode

1.1. Clone and Initialize the Repository

Begin by creating or cloning your repository on GitHub. In this case, clone from https://github.com/kyleengza/pi-live-detect-rstp.
From your command line:

```
git clone https://github.com/kyleengza/pi-live-detect-rstp.git
cd pi-live-detect-rstp
```

Rationale: Cloning the project ensures you start with the latest version, which may already include initial structure and documents such as the project plan, modularization notes, and service outlines.

1.2. Open in VSCode

Open the repository folder in VSCode:

```
code .
```

This allows you to leverage all of VSCode’s capabilities-navigation, search, refactoring, Copilot integration, and source control-directly in your development environment.

1.3. Remote Development Setup for the Raspberry Pi 5

Your Raspberry Pi 5 is the target runtime. While development can occur on your main machine, you will ultimately need to test and run code natively.

Recommended Approach: Use the VSCode Remote-SSH extension to develop on the Pi itself from your main desktop or laptop[2][3]. This approach maintains the benefit of your local tooling and interface while ensuring compatibility with the Pi's ARM64 environment and immediate access to Pi-specific libraries and hardware.

Steps:

1. Ensure SSH is enabled on the Pi (sudo raspi-config > Interface Options > SSH: Enable).
2. Install VSCode’s Remote-SSH extension.
3. In VSCode, use the “Remote Explorer” to add a new Pi connection (ssh pi@<pi_ip_address>), then open a new VSCode window in remote session.
4. All subsequent actions-file editing, Copilot, and terminal access-will operate remotely on the Pi, with a seamless local experience.

Benefits: This method avoids most cross-platform pitfall, such as dependencies only available for arm64/Linux, and gives you direct access to the Pi’s hardware and Docker (if required)[1][3].

1.4. Project Structure & Initial Scaffolding

A modular microservices architecture benefits from a clear folder and file structure. You can leverage Copilot’s workspace creation capabilities to scaffold the project, especially using Copilot Chat’s /createWorkspace or through explicit prompts.

A recommended structure for this RTSP pipeline:

```
pi-live-detect-rtsp/
□
□ □ services/
□ □ □ stream_probe/
□ □ □ frame_collector/
□ □ □ image_detection/
□ □ □ annotation/
□ □ □ rtsp_annotator/
□
□ □ shared/
□ □ □ database/
□ □ □ logging/
□ □ □ utils/
□
□ □ tests/
```

```
│
├── docs/
│
├── config/
│
├── requirements.txt
├── README.md
├── .gitignore
├── docker-compose.yml (optional)
├── .github/
│   └── workflows/
```

Using Copilot for Custom Scaffolding: In Copilot Chat you can enter:

/createWorkspace
A modular Python project for Raspberry Pi 5 and Hailo8L. Services include stream probing, frame collection/caching, TF + YOLO detection with Hailo8L, annotation, and RTSP restreaming. Use an in-memory database and text-based logging. Create folders for each service, shared utilities, docs, tests, and CI workflows. Add README and requirements.txt.

Outcome: Copilot will propose a full structure and optionally create it for you, populating minimal viable code in each folder. However, always review generated scaffolding for alignment with the architectural vision and make tweaks as necessary.

---

2. Configuring GitHub Copilot for Python in VSCode

2.1. Install Copilot Extension

- From the Extensions sidebar in VSCode, search for “GitHub Copilot” and install it[4].
- Also, install the “Python” extension by Microsoft, essential for Python code intelligence and linting[4].

Tip: Ensure you are signed into VSCode’s GitHub account with an active Copilot subscription or the Copilot Free plan[6][7].

2.2. Configuring Copilot Settings

- From the Command Palette (Ctrl+Shift+P or Cmd+Shift+P), open GitHub Copilot: Sign In if not signed in.
- Set Copilot autocomplete level, code suggestion frequency, and chat suggestion options via the “Copilot” settings in VSCode’s preferences[8].

Adjust your workspace Copilot settings for the project. For example, you can limit suggestions to Python files, enable or disable telemetry, or select the AI model used.

2.3. Write Effective Copilot Instructions Files

Maximizing suggestion quality starts with project context. Create a copilot-instructions.md file in your repo root (or .github/ as copilot-instructions.md). This file should give Copilot a mini-spec for what the code should do, technology stack, style preferences, library constraints, and specific examples[10]. This directly influences code completions and Copilot Chat output for your project.
Suggested content for a detection project:

- **Project overview:** Explain the overall RTSP pipeline, modularity, target hardware, and expected performance.
- **Tech stack and constraints:** Specify Python 3.11+, TensorFlow + YOLO (Hailo-8L optimized), opencv-python, PIL, Redis (if used), logging standards, and ARM64 Linux compatibility.
- **Coding guidelines:** PEP8, docstrings for public functions, use dataclasses, etc.
- **Service boundaries:** Describe inter-service communication and use of in-memory DB.

Example Copilot instruction prompt:

```
This Python project targets Raspberry Pi 5 with Hailo8L. It consists of modular microservices for:
- Probing and ingesting RTSP video
- Collecting/caching decoded frames in-memory
- Performing AI inference with TensorFlow + YOLO (optimized for Hailo8L) on incoming frames
- Annotating frames with bounding boxes and class labels
- Restreaming annotated frames via RTSP
Frameworks: TensorFlow (with HailoRT/pyHailoRT for acceleration), Redis or in-memory
structures for cross-service data, text-based persistent logging, and Linux compatibility. Use
clear separation of concerns, type-hints, PEP8 and modular code.
```

Why this matters: Richer instructions = Copilot suggestions that fit your actual needs and project infrastructure[10].

2.4. Copilot Keyboard Shortcuts and Usage

Familiarize yourself with VSCode and Copilot keyboard shortcuts:

- **Tab:** Accept inline suggestion
- **Ctrl+Alt+I (Cmd+Opt+I):** Open Copilot Chat
- **Ctrl+Shift+B:** Build task (useful if you set up testing/building)
- **Ctrl+Shift+X:** Extensions sidebar

You can ask Copilot Chat for code explanations, function generation, regexes, code reviews, refactoring suggestions, and even documentation[11][12].

---

3. Service Scaffolding: Modular Python Microservices

3.1. Design Modular Services

Based on your project plan, each significant function is implemented as a standalone service, possibly running in a separate process (or container). Each service is responsible for a focused task and communicates through shared memory objects or an in-memory database.
Key services:

- **Stream Prober:** Validates and maintains the RTSP source connection.
- **Frame Collector:** Decodes, collects, and caches video frames.
- **Image Detection:** Executes object detection on frames with specialized hardware acceleration.
- **Annotation:** Draws bounding boxes and class labels on frames.
- **RTSP Restreamer:** Streams the annotated frames out to clients.

3.2. Use Copilot to Scaffold Each Service

Approach:
For each service:

1. Create a dedicated subdirectory with __init__.py, core service Python file(s), and any configs.
2. Start each file with a clear docstring and, optionally, a function/class-level description. This guides Copilot in producing relevant boilerplate code.

For example, create services/frame_collector/frame_collector.py and add:

```
"""
Frame Collector Service

- Connects to the source RTSP stream.
- Decodes video and extracts frames.
- Stores frames in an in-memory cache or Redis.
- Publishes frame metadata for downstream services.
"""
```

Begin coding by writing a function signature or a natural language comment. Copilot will suggest code completion inline or in Copilot Chat. Accept, modify, or reject suggestions based on accuracy and completeness.
Prompt example for Copilot:

```
# Create a class FrameCollector that uses OpenCV to connect to an RTSP stream, extracts decoded frames, and caches latest 10 frames in a ring buffer shared with other microservices.
```

Tip: Be as explicit as possible-number of frames, buffering strategy, and class names help Copilot generate targeted, high-fidelity code.

3.3. Project Structure Table

Service Name	Directory	Key Responsibilities	Main Technologies
Stream Prober	services/stream_probe/	Check RTSP endpoint availability, handle retries	OpenCV, requests
Frame Collector	services/frame_collector/	Connect to RTSP, decode and cache video frames	OpenCV, numpy, Redis
Image Detection	services/image_detection/	Runs YOLO/TensorFlow models, Hailo8L hardware acc.	TensorFlow, HailoRT
Annotation	services/annotation/	Draws bounding boxes and labels on images	PIL, OpenCV, TensorFlow
RTSP Restreamer	services/rtsp_annotator/	Streams annotated frames to an RTSP server	Zephyr-rtsp, ffmpeg
Shared DB	shared/database/	Manages in-memory cache, cross-service synchronization	Redis, dict, queue
Logging	shared/logging/	Handles text-based logs for all services	Python logging

Analysis:
Each directory separates code, configurations, and dependencies, promoting independent testing, scalability, and future containerization/deployment. Shared utilities (in-memory database, logging, and common helpers) maximize code reuse.

---

4. Service Implementation with Copilot Assistance

4.1. Probing RTSP Streams

OpenCV Method (Python Example):

```
import cv2

class StreamProber:
def __init__(self, rtsp_url):
self.rtsp_url = rtsp_url

def probe(self):
cap = cv2.VideoCapture(self.rtsp_url)
if not cap.isOpened():
# Log error
return False
ret, frame = cap.read()
```

cap.release()
return ret

Copilot Prompt Example:

# Implement an RTSP stream checker. Try connecting using OpenCV, include retries, exponential backoff, and logging for each attempt.

Review Copilot's output for connection handling and error management, checking codec compatibility and stream stability[13].
Alternative: Zephyr-rtsp or python-rtsp-server: Consider using Zephyr-rtsp or python-rtsp-server for advanced needs such as proxying or aggregating multiple RTSP streams[15].

4.2. Frame Collection and In-Memory Caching

Caching Strategies:

- Built-in Python dict or deque for simple, local memory cache.
- Redis for high-performance, cross-process or distributed cache (strongly recommended for microservice architectures).

Copilot Prompt Example:

# Implement a FrameCache class that stores latest N frames in memory, thread-safe, with set/get methods for frame retrieval.

Analysis:

- Specify thread/process safety if relevant, as multiple processes may access frames for detection and annotation.
- Redis is ideal for in-memory syncing, as it's widely used, robust, and has good Python bindings (redis-py)[17][18].

Pro tip: Use channels/streams in Redis or pub/sub to synchronize new frames/events across services for fine-grained orchestration.

4.3. Detection Service: Running YOLO/TensorFlow Optimized for Hailo-8L

Setup Steps:

- **Install Hailo drivers and pyHailoRT/TensorFlow**. On Raspberry Pi: sudo apt install hailo-all [20].
- Ensure PCIe Gen3 is enabled in firmware for maximum performance.
- Copy pre-compiled YOLO models as .hef files, obtained from Hailo Dataflow Compiler on x86, due to unsupported compilation on ARM[21].

Copilot Prompt Example:

# Implement Detector that loads a YOLO HEF model for Hailo8L and runs inference on in-memory frames, returning bounding boxes and class labels.

Key notes:

- Use sample code from Hailo's official rpi5-examples and DeGirum's PySDK samples as inspiration[23].
- Leverage GStreamer and hardware-accelerated pipelines for efficiency-Python APIs tend to use more CPU, so use Hailo's recommended approach when possible for post-processing[23].

Copilot's strengths here: Turn partial sample code or interface documentation into tailored detection service code for your specific model/class/label layout.

4.4. Annotating Frames

- Use OpenCV or TensorFlow utilities (e.g., tf.image.draw_bounding_boxes) to draw on raw frames with detection results[25][26].
- Ensure annotations are visually prominent but non-obstructive-Copilot can create generic drawing routines with descriptive prompts.

Copilot Prompt Example:

# Write a function that draws bounding boxes and class labels on an image, using color coding for each class. Frame shape is numpy array.

Tailor to your color scheme, label font, and output frame format.

4.5. Re-Streaming Annotated Frames over RTSP

Implementation options:

- Use Zephyr-rtsp's Stream API to send frames to a local RTSP server (e.g., MediaMTX or python-rtsp-server)[14].
- Alternative: use FFmpeg via subprocess to restream video-Copilot can generate invocation code for this.

Copilot Prompt Example:

# Implement a class that takes annotated frames (numpy images) and streams them over RTSP via Zephyr-rtsp to rtsp://localhost:8554/stream.

Coordinate output stream resolution, framerate, and codec for compatibility with consuming clients.

4.6. Service Orchestration with In-Memory Database

- Use Redis (or Python's multiprocessing/manager dict for small setups) as a transient data/message bus.
- Pub/sub and streams enable scalable, resilient orchestration.

- Each service publishes/consumes messages about frame availability, detection results, error state, health-checks.

Copilot Prompt Example:

# Set up a service discovery and coordination pattern using Redis. Each service announces its status and processes a queue of events from other services.

This enables decoupling, health monitoring, and performance scalability[18][16].

4.7. Logging to Text Files in Modular Services

- Use Python's logging module, NOT print statements-configure a logger per service, with rotating file handlers for robust, parseable logs[30][31].
- Copilot can suggest best-practice logger configuration and custom formatters for each module.

Copilot Prompt Example:

# Configure per-service Python logger. Log to text file with timestamps, service name, log level, and rotating files at 5 MB per file.

Proper logging ensures traceability and simplifies debugging and ops.

---

5. Testing Python Microservices on Raspberry Pi

5.1. Use Pytest and Copilot for Automated Unit and Integration Tests

- Scaffold tests in a tests/ directory, grouping by service.
- Use Copilot Chat to generate parameterized test cases and mock data.

Copilot Prompt Example:

# Generate pytest unit tests for FrameCollector, including tests for valid RTSP stream, invalid URL, reconnection, and empty frames.

- Ask Copilot to explain test structure or edge-case coverage for robust test generation[6].

Tip: For hardware-in-the-loop/embedded edge cases, combine with tools like pytest-embedded for device testing.

5.2. Test on Real or Emulated Pi Environment

- Use VSCode's integrated terminal (connected over Remote-SSH) to run your test suites natively on the Pi, ensuring dependency compatibility.
- For CI, consider adding self-hosted runners running on the Pi for action-based continuous testing[33].

---

6. Documentation Generation with VSCode and Copilot

6.1. Docstrings and Inline Comments

- As you write or refactor functions, Copilot can auto-complete docstring templates based on signatures and usage.
- Use Copilot Chat: “Add docstrings for all public functions in this file.”

6.2. Project-Level Documentation

README/overview generation:

- Use Copilot Chat’s /create-readme or similar instructions in a blank README.md to auto-generate initial project documentation based on project context and code structure[35].
- Review, revise, and add explicit usage examples, setup steps, and diagrams as needed[36].

Prompt for Copilot:

# Generate detailed README.md for a modular RTSP detection and annotation system running on Raspberry Pi 5 with Hailo8L, including installation, usage, architecture diagram (Markdown), service descriptions, and contact info.

6.3. API and Code Documentation

- Copilot can write OpenAPI spec snippets or Sphinx/reStructuredText for documentation generation, which is especially helpful for REST interfaces or serialized message schemas.

---

7. GitHub Integration and CI/CD for Raspberry Pi Projects

7.1. Using GitHub for Version Control

- Use built-in VSCode Git tools: source control sidebar for commits, branching, merges.
- Copilot can generate informative commit messages-e.g., “fix(image_detection): optimize YOLO inference for Hailo8L hardware”[37].

Best practices:

- Follow semantic/conventional commit conventions for clarity and automated changelog generation.
- Make frequent, atomic commits to facilitate code reviews and traceability.

7.2. Setting Up CI/CD for ARM (Raspberry Pi)

Most cloud-hosted runners are x86-based and may not suffice if native hardware testing is required.

- Option 1: **Self-hosted runners**-deploy a GitHub self-hosted runner service on your Pi, enabling action workflows (testing/build, even deployment) locally[33].
- Option 2: **Hybrid**-run lint/type-check/unit tests on x86 GitHub runners; only run hardware-in-the-loop/integration on Pi.

Example GitHub Actions workflow for Python:

```
name: Python CI

on:
push:
branches: [ main ]
pull_request:
branches: [ main ]

jobs:
build-and-test:
runs-on: ubuntu-latest
steps:
- uses: actions/checkout@v4
- name: Set up Python
uses: actions/setup-python@v5
with:
python-version: '3.11'
- name: Install dependencies
run: |
python -m pip install --upgrade pip
pip install -r requirements.txt
- name: Run tests
run: |
pytest
```

For integration with your Raspberry Pi, use the same YAML but set runs-on: self-hosted and deploy the runner service on the Pi[32].

7.3. Using Copilot for Commit Messages and Pull Requests

- In VSCode, select staged changes and use Copilot to generate descriptive commit messages and PR summaries.
- You can prompt Copilot for "quick summary of changes" or "PR template for feature addition X."
- Reviewing code: use Copilot Chat for "review this file for edge case bugs" or "suggest test coverage improvements."

8. Tips for Maximizing Copilot Usefulness and Avoiding Pitfalls

8.1. Copilot Prompt Engineering

- **Be explicit:** State libraries, target hardware, input/output data shapes, and expected side effects. For example, “Write a YOLOv8 postprocessing function compatible with Hailo8L and returns [class, confidence, bbox] for each object.”
- **Break complex tasks:** Larger prompts for high-level structure, small prompts for focused code generation.
- **Iterate:** Accept, review output critically, refine, or reject. Copilot is not infallible-adjust for correctness and edge-case coverage[12].

8.2. Project-Wide Context for Copilot

- Use copilot-instructions.md and well-structured, documented code/files. Copilot leverages open files, so the more targeted your workspace, the better the results.

8.3. Validation and Refactoring

- Always review code for correctness, security, and hardware/resource compatibility-especially when dealing with concurrency, I/O, or hardware-accelerated libraries.
- Ask Copilot to explain code if unsure, or to refactor for readability/resilience.

8.4. Avoiding Common Pitfalls

- Don’t blindly trust code that manages resources or safety-critical IO (network, threading, hardware)-always validate, write tests, and review Copilot’s output.
- Explicitly test Copilot-generated code for error handling, resource cleanup, and security in all services.
- Watch for overfitting to similar patterns-refresh instructions or prompts if output is too generic or stale.

Copilot Usefulness Table

Area	How Copilot Excels	Pitfalls to Avoid	Recommendations
Scaffolding	Proposes modular folders, setup files, stubs	May misalign directory for special hardware/project edge cases	Always review vs. project plan
Function Coding	Boilerplates, interface generation, docstrings	Logic errors, partial coverage	Manual test and code review

Tests	Generates unit/integration tests from specs	May miss hardware dependencies, external resources	Adapt test to hardware, refactor
Documentation	README, inline docstring auto-generation	Overly generic, outdated if code changes	Supplement with manual sections
Commit Messages	Contextual commit messages/PR summaries	Vague messages if stage changes are insufficient	Add manual review

---

Conclusion

Using Visual Studio Code and GitHub Copilot on a Raspberry Pi 5 with Hailo8L, you can build a resilient, high-performance modular RTSP detection and annotation pipeline by adhering to a disciplined workflow. Key steps-remote development, Copilot context, modular service scaffolding, code/test generation, and actionable documentation-ensure quality, maintainability, and fast iteration. Take advantage of Copilot's prompt sensitivity, rich context, and chat tools to accelerate every phase, but always validate and test for the project's unique demands. Structure, clarity, and targeted Copilot usage are your strongest allies in delivering a robust, edge-optimized vision AI solution.

References (37)

1. *Programming Raspberry Pi Remotely using VS Code (Remote-SSH)*. https://randomnerdtutorials.com/raspberry-pi-remote-ssh-vs-code/
2. *GitHub - seapanda0/hailo-yolo-guide: Guide to Convert and Inference …*. https://github.com/seapanda0/hailo-yolo-guide
3. *Benchmark on RPi5 & CM4 running yolov8s with Hailo 8L*. https://forums.raspberrypi.com/viewtopic.php?t=373867
4. *Object detection: Bounding box regression with Keras, TensorFlow, and …*. https://pyimagesearch.com/2020/10/05/object-detection-bounding-box-regression-with-keras-tensorflow-and-deep-learning/
5. *python - How to use boundary boxes with images for multi label image …*. https://stackoverflow.com/questions/58238146/how-to-use-boundary-boxes-with-images-for-multi-label-image-training
6. *10 Best Practices for Logging in Python - Better Stack*. https://betterstack.com/community/guides/logging/python/python-logging-best-practices/
7. *Python Logging Best Practices (with 12 Code Examples) - SigNoz*. https://signoz.io/guides/python-logging-best-practices/

8. *Continuous Integration and Deployment for Python With GitHub Actions*. https://realpython.com/github-actions-python/
9. *Using my new Raspberry Pi to run an existing GitHub Action*. https://blog.frankel.ch/raspberry-pi-github-action/
10. *AI Kit and AI HAT+ software - Raspberry Pi Documentation*. https://www.raspberrypi.com/documentation/computers/ai.html
11. *Lesson 4: Creating Documentation with Copilot - GitHub*. https://github.com/neudesic/learning-github-copilot/blob/main/docs/4-creating-documentation-with-copilot.md
12. *Generate Documentation Using GitHub Copilot Tools - Training*. https://learn.microsoft.com/en-us/training/modules/generate-documentation-using-github-copilot-tools/
13. *Using VS Code for Python Development with Remote RPi5 Access*. https://circuitlabs.net/using-vs-code-for-python-development-with-remote-rpi5-access/
14. *Coding on Raspberry Pi remotely with Visual Studio Code*. https://www.raspberrypi.com/news/coding-on-raspberry-pi-remotely-with-visual-studio-code/
15. *Get Started with GitHub Copilot with VSCode and Python Extension*. https://techcommunity.microsoft.com/blog/educatordeveloperblog/get-started-with-github-copilot-with-vscode-and-python-extension/3736564
16. *Build with Redis data structures for microservices using Amazon ...*. https://aws.amazon.com/blogs/database/build-with-redis-data-structures-for-microservices-using-amazon-memorydb-for-redis-and-amazon-ecs/
17. *Quickstart for GitHub Copilot*. https://docs.github.com/en/copilot/get-started/quickstart?tool=vscode
18. *GitHub Copilot in VS Code cheat sheet - Visual Studio Code*. https://code.visualstudio.com/docs/copilot/reference/copilot-vscode-features
19. *vscode-docs/docs/copilot/reference/copilot-vscode-features.md ... - GitHub*. https://github.com/microsoft/vscode-docs/blob/main/docs/copilot/reference/copilot-vscode-features.md
20. *GitHub - fielding/copilot-instructions: A curated list of code ...*. https://github.com/fielding/copilot-instructions/
21. *Code completions with GitHub Copilot in VS Code*. https://code.visualstudio.com/docs/copilot/ai-powered-suggestions
22. *GitHub Copilot: Complete Guide to Features, Limitations ... - Swimm*. https://swimm.io/learn/github-copilot/github-copilot-complete-guide-to-features-limitations-alternatives
23. *capturing rtsp camera using OpenCV python - Stack Overflow*. https://stackoverflow.com/questions/40875846/capturing-rtsp-camera-using-opencv-python
24. *Lightweight, zero-dependency proxy and storage RTSP server*. https://github.com/vladpen/python-rtsp-server
25. *GitHub - RedisGears/EdgeRealtimeVideoAnalytics: An example of using ...*. https://github.com/RedisGears/EdgeRealtimeVideoAnalytics

26. *zephyr-rtsp · PyPI*. https://pypi.org/project/zephyr-rtsp/
27. *In-Memory Databases that Work Great with Python - DEV Community*. https://dev.to/memgraph/in-memory-databases-that-work-great-with-python-553m
28. *Conventional Commits Cheatsheet · GitHub*. https://gist.github.com/qoomon/5dfcdf8eec66a051ecd85625518cfd13