---

# User Guide for Linux-Driver-Code-Grader

This document explains how to install, run, and analyze evaluations of Linux driver source files using the **Linux-Driver-Code-Grader** system.

---

### 1. Prerequisites

The evaluator requires a Linux environment with kernel headers. Supported environments:

- **Ubuntu 22.04+** (recommended)
- **GitHub Codespaces** (tested)
- **Other Linux distributions** with minimal adjustments

**Required Packages**

- Python 3.9+
- build-essential (gcc, make)
- Linux kernel headers (/lib/modules/$(uname -r)/build)
- Static analysis tools:
    - sparse
    - clang-format, clang-tidy
    - cppcheck
- gdb
- kmod (for insmod, rmmod, lsmod)

---

### 2. Installation

**Clone the repository**

git clone https://github.com/VedantNipane/Linux-Driver-Code-Grader.git

cd Linux-Driver-Code-Grader

**Setup environment**

The repo provides a helper script:

chmod +x setup.sh

./setup.sh

This installs all necessary dependencies (requires sudo).

---

**3. Running Evaluations**

**Evaluate a single driver**

python3 Evaluator/evaluator.py Tests/sample_driver.c

This will:

1. Attempt compilation (kbuild first, fallback to gcc).
2. Run static analysis (style, parser, security, performance).
3. Run dynamic tests (load/unload, dmesg checks, smoke tests).
4. Generate a JSON report under outputs/ and append results to score_logs.csv.

---

**Run evaluation on all test drivers**

Use the provided Makefile:

make run

This executes evaluator.py against every file in Tests/ and collects outputs.

---

**4. Understanding Outputs**

**Console Report**

Each run prints a structured report:

=== Evaluation Report ===

File: Tests/sample_driver.c

Compilation: Success (method=kbuild)

Warnings: 1 Errors: 0

--- Score Breakdown ---

Correctness: 38.0/40

- Compilation: success (method=kbuild)
- Functionality score: 1.00 -> 10.00/10
- Runtime score: 8.0/10 (compiled, loaded, unloaded)

Security: 23.1/25

- sub_scores: {memory_safety: 0.7, resource_mgmt: 1.0, ...}

- issues: ['unsafe_function:strcpy']

Code Quality: 13.2/20

Performance: 8.0/10

Advanced: 0.0/5

Overall Score: 82.3/100

Report saved to: outputs/sample_driver_results.json

---

**JSON Report**

Stored under outputs/filename_results.json.
Contains structured data:


```
{
  "file": "Tests/sample_driver.c",
  "overall_score": 82.3,
  "breakdown": {
    "Correctness": {"awarded": 38.0, "max": 40, "details": [...]},
    "Security": {"awarded": 23.1, "max": 25, "details": [...]}
  },
  "compilation": {...},
  "runtime": {...},
  "security": {...},
  "style": {...},
  "performance": {...}
}
```


This format is machine-readable for downstream analysis.

---

**CSV Log**

All evaluations append results to score_logs.csv (repo root).
Example entry:

timestamp,file,overall_score,Correctness,Security,Code Quality,Performance,Advanced

2025-09-18 17:42:03,sample_driver.c,77.5/100,34.0/40,18.8/25,16.7/20,8.0/10,0.0/5

Useful for tracking trends across multiple runs.

---

**5. Technical Details**

**Compilation Modes**

- **kbuild (preferred):** Uses Linux kernel's native build system. Produces .ko modules if successful.
- **gcc fallback:** Syntax-only check if headers are missing or kbuild fails. May yield a "soft pass".

**Dynamic Tests**

- Module insertion/removal via insmod / rmmod
- Kernel log check via dmesg
- Smoke test for device node presence in /dev or /proc/devices
- Output integrated into **Correctness score**

**Scoring**

- Correctness capped at 40 (compilation + structure + runtime)
- Security normalized from multiple sub-checks
- Performance starts at full score, deductions applied
- Advanced features (devm, device tree, pm hooks, debugfs) give bonus points

---

**6. Limitations**

- Requires root/sudo for runtime tests (insmod, rmmod).
- Kernel headers must match the running kernel version.
- Smoke tests do not validate driver functionality beyond load/unload and node creation.
- Performance heuristics are static (no stress testing).

---

**7. Next Steps**

- Extend dynamic tests (I/O validation, stress tests).
- Broader driver type support (block, network).
- Integration with CI pipelines for automated evaluation.

---