Rubrics Explanation

The evaluation rubric defines how driver submissions are scored across multiple technical dimensions. Each dimension has a specific weight, reflecting its importance in the overall quality of a Linux kernel driver. Scores are normalized to a scale of 100.

---

1. Category Weights

Category	Weight	Purpose
Correctness	40%	Ensures the driver builds, integrates with the kernel, and performs basic runtime functions.
Security	25%	Verifies safe handling of memory, resources, and synchronization to prevent kernel vulnerabilities.
Code Quality	20%	Enforces Linux coding standards, maintainability, and documentation to ensure long-term usability.
Performance	10%	Evaluates efficiency and scalability of driver operations, avoiding resource-heavy patterns.
Advanced	5%	Rewards use of advanced kernel features such as device tree support, power management, and debugging hooks.

The weighting reflects a balance between **must-have criteria** (correctness, security) and **nice-to-have criteria** (advanced features).

---

2. Detailed Breakdown

A. Correctness (40%)

Sub-metric	Max Score	Rationale
Compilation Success	30	A driver must compile cleanly with the kernel build system (kbuild) to be functional. Errors are fatal. If compilation fails only due to missing headers (e.g., no kernel sources), a "soft pass" is awarded.
Structural Functionality	10	Assesses whether required callbacks and functions are implemented for the driver type. For example, a character

Sub-metric	Max Score	Rationale
		device should define open, read, write, and release. Missing callbacks reduce functional completeness.
Runtime Behavior	Up to 10 (within 40)	Additional sub-score that validates the driver at runtime: module load/unload, dmesg logging, and smoke tests. This is capped such that Correctness never exceeds 40 in total.

Rationale: Correctness is weighted highest because a driver that cannot compile, load, or function correctly is unusable.

B. Security (25%)

Sub-metric	Max Score	Rationale
Memory Safety	0–1 (normalized)	Detects use of unsafe functions (strcpy, unchecked copy_from_user, etc.) and improper buffer handling. Kernel memory bugs often lead to privilege escalation.
Resource Management	0–1	Ensures proper allocation/release of memory and device resources. Prevents leaks and dangling pointers.
Concurrency / Race Conditions	0–1	Checks synchronization primitives (mutex, spinlock, atomic ops). Absence of proper locking can corrupt shared state in kernel context.
Input Validation	0–1	Validates that user-provided data is sanitized before use (length checks, bounds checking). Protects against malformed or malicious inputs.

The average of sub-scores is scaled to 25.

Rationale: Security is prioritized heavily because kernel code executes with full system privileges, and unsafe drivers compromise system integrity.

C. Code Quality (20%)

Sub-metric	Max Score	Rationale
Style Compliance	40% of 20	Uses checkpatch.pl and heuristics to check adherence to the Linux coding style. Consistency improves readability and maintainability.

Sub-metric	Max Score	Rationale
Documentation	30% of 20	Ensures presence of kernel doc comments, inline explanations, and clarity of exported symbols. Aids maintainers and future contributors.
Maintainability	30% of 20	Measures simplicity, modularity, and avoidance of anti-patterns. Drivers with maintainable structure are easier to extend and debug.

Rationale: While correctness and security are mandatory, quality ensures the driver is maintainable and aligned with upstream standards.

D. Performance (10%)

Sub-metric	Max Score	Rationale
Complexity	Penalty-based	Penalizes unnecessarily high cyclomatic complexity in driver callbacks, which can affect responsiveness.
Memory Usage	Penalty-based	Flags oversized kmalloc allocations or inefficient memory management.
Scalability	Penalty-based	Detects blocking operations or patterns that would not scale under load.

Rationale: Drivers should be efficient to avoid degrading kernel performance. However, this is weighted less heavily since performance optimization is secondary to correctness and safety.

E. Advanced Features (5%)

Feature	Max Award	Rationale
Managed Resource Allocation (devm_*)	1.5	Encourages modern kernel APIs that simplify resource cleanup.
Device Tree Support (of_match_table)	1.5	Essential for embedded and ARM platforms where device trees are standard.
Power Management Hooks (suspend, resume, pm_ops)	1.0	Extends driver functionality for mobile and power-sensitive environments.

Feature	Max Award	Rationale
Debugging Support (debugfs, pr_debug)	1.0	Facilitates kernel debugging and diagnosis.

Rationale: Advanced features are rewarded to distinguish drivers that go beyond the basics. These features demonstrate maturity and readiness for upstream contribution.

---

3. Scoring Approach

- **Normalization:** Each sub-score is normalized to its category weight.
- **Soft Passes:** If missing kernel headers prevent a full build, syntax-only compilation may still award partial correctness points.
- **Penalty Model:** For performance, scores start at full marks (10/10) and deductions are applied for detected inefficiencies.
- **Caps:** Correctness cannot exceed 40 even with runtime checks; runtime is a component, not an extra bonus.

---

4. Summary

The rubric ensures a **balanced evaluation**:

- Critical dimensions (**correctness, security**) dominate the score.
- Supporting dimensions (**quality, performance**) ensure long-term value.
- Advanced features provide **bonus recognition** for completeness.

This structure aligns with both academic evaluation (clear scoring criteria) and real-world kernel development practices (emphasis on correctness and safety).
