ChatGPT

Kernel/OS Functionality Scoring Rubric

This rubric defines a **standardised metric** for evaluating how well a software repository implements core **kernel** and **operating-system (OS)** primitives. It is based on the function manifest and status report from the Echo.Kern project and draws on general operating-system principles [1]. The goal is to provide a repeatable method for scoring any repository – regardless of its intended purpose – so we can measure whether it contains kernel/OS functionality and how complete those features are relative to the expectations of an AGI-ready kernel.

1 Design principles

1. **Use a canonical feature list.** The rubric draws on the 10 core kernel primitives identified in the Echo.OCC evaluation (boot/init, scheduling, process management, memory management, interrupt handling, system calls, basic I/O, synchronisation primitives, timers/clock and protection/privilege separation) [1]. These are supplemented with platform-level OS services (virtual memory, device drivers, filesystem, networking, inter-process communication, security subsystems, power management and profiling) found in the Echo.Kern function manifest [1] and status report.

2. **Weight by importance.** Each feature is assigned a weight reflecting its criticality. Core kernel primitives receive higher weights than platform features. Within the core, boot/initialisation and CPU scheduling carry the highest weight because no kernel can operate without them, while timers/clock have a lower weight. Platform services receive moderate weights because they are needed for a full OS but not to boot a minimal kernel.

3. **Measure both presence and completeness.** A repository may expose some primitives but only partially implement them. To capture this nuance, each category's score is the product of:

4. **Presence:** whether any evidence of the feature exists (binary or graded by textual/functional evidence).
5. **Completeness:** how fully that feature is implemented. The manifest and status report define functions and target SLOC; the fraction of those present in the repository determines completeness. If the repository implements 3 of 10 required functions in a category, completeness is 0.30.

6. **Weight:** the category's importance multiplier.

7. **Total scores normalised to 100.** Kernel and OS scores are normalised separately. A high kernel score indicates the repository contains significant kernel primitives; a low score suggests it is application-level or "other".

2 Kernel primitives and weights

The table below lists the **core kernel primitives**, summarises evidence to look for, and assigns a **weight** (total weight = 60). The evidence column suggests keywords or files one might search for in a

repository. The target number of functions is taken from the Echo.Kern manifest and status report [1]; these numbers provide the basis for the completeness ratio.

Primitive	Weight	Evidence/keywords	Manifest target	Notes
Boot / initialisation	**10**	files like `boot.c`, `stage0_bootstrap`, `init_membranes`, assembly boot code	12 functions (~5 k SLOC)	Must bring CPU/ memory to known state [1].
CPU scheduling	**9**	`scheduler.c`, `sched_*` functions, context switch routines	18 functions (~8 k SLOC)	Includes tick handler and runqueue management [1].
Process/thread management	**8**	functions to create/destroy processes/threads, `fork`, `spawn`, `thread_init`	24 functions (~N/A)	May overlap with scheduler; presence of thread structures and context management.
Memory management	**8**	`malloc`, `memory.c`, `alloc_page`, `free`, hypergraph allocator	24 functions (~12 k SLOC)	Basic heap/stack plus virtual memory support [1].
Interrupt handling & traps	**6**	`interrupt.c`, `vector_table`, `IRQ_handler`, trap stubs	15 functions (~6 k SLOC)	Handles hardware interrupts and synchronous faults.
System call interface	**5**	`syscalls.c`, syscall table, ABI entry points	32 functions (~10 k SLOC)	Gateway between user mode and kernel services [1].
Basic I/O primitives	**5**	`io.c`, device register access, `read/write` low-level operations, HALs	20 functions (~7 k SLOC)	Minimal means to talk to devices or hardware ports.
Low-level synchronisation	**4**	`spinlock`, `mutex`, `atomic`, `barrier` functions, lock structs	16 functions (~4 k SLOC)	Required for safe concurrency.
Timers and clock	**3**	`timer.c`, tick counter, `clock_gettime`, scheduling quantum	10 functions (~3 k SLOC)	Provides timekeeping and periodic interrupts.
Protection / privilege separation	**2**	`mmu`, `protect`, `privilege`, `user/kernel mode switch`	14 functions (~6 k SLOC)	Ensures isolation and privilege boundaries.

Scoring calculation for a primitive

For each primitive i, calculate:

```
presence_i ∈ {0, 0.5, 1}  – evidence of any relevant code (0 = none,
0.5 = superficial mention, 1 = substantial implementation)
completeness_i = (#functions found in repo) / (target functions in manifest)
score_i = weight_i × presence_i × completeness_i
```

Functions can be counted using static analysis: search for function prototypes matching names in the manifest or measure lines of code in relevant files. If static analysis isn't available, approximate completeness by the proportion of required sub-modules present (e.g., if `scheduler.c` exists but `sched_policy.c` and `scheduler.h` are missing, completeness ≈ ⅓). Sum all `score_i` across the 10 primitives. Finally, normalise the kernel score:

```
kernel_score = (Σ score_i) / (maximum possible score = Σ weights)
× 100
```

3 Operating-system (platform) services

Beyond the core kernel, a full operating system offers higher-level services. These categories use lower weights (total weight = 40) and can be scored similarly. A high OS score indicates the repository contains substantial platform code (drivers, filesystems, networking, etc.).

Service	Weight	Evidence/keywords	Manifest target	Notes
Virtual memory / paging	**8**	`page_table`, `virtual memory`, `mmu`, `paging.c`	~28 functions (~15 k SLOC)	Implements address translation and memory protection.
Device driver framework	**8**	`driver`, `register_driver`, bus abstractions, HAL stubs	~35 functions (~20 k SLOC)	Framework for attaching devices; includes neuromorphic HAL [1].
Filesystem / VFS	**7**	`fs.c`, `vfs`, `open`, `read/write`, inode structures	~42 functions (~25 k SLOC)	A virtual file system layer over a hypergraph FS.
Networking stack	**5**	`socket`, `network.c`, `tcp`, `udp`, driver glue	~58 functions (~35 k SLOC)	May be absent in embedded kernels.

Service	Weight	Evidence/keywords	Manifest target	Notes
Inter-process communication (IPC)	**4**	`message_queue`, `pipe`, `signal`, `psystem_membranes`	~18 functions (~10 k SLOC)	Includes message passing and P-system membranes.
Security subsystems	**3**	`crypto`, `authentication`, `capability`, `attestation`	~30 functions (~18 k SLOC)	Includes cryptographic attestation used in Stage0 boot [1].
Power management	**3**	`power.c`, `sleep`, `wake`, energy budgets	~22 functions (~12 k SLOC)	Optional for embedded or research kernels.
Profiling & debug	**2**	`profiler.c`, `trace`, `debug`, performance counters	~25 functions (~15 k SLOC)	Tools to monitor and tune performance.

Compute each service's score with the same formula as core primitives and normalise to obtain an **OS score** (0–100). When calculating the completeness ratio, use the target SLOC from the manifest or status report as the denominator; count SLOC in the repository's corresponding modules as the numerator.

4 Classification and interpretation

After computing **kernel_score** and **os_score**, classify the repository:

Classification	Criteria	Interpretation
Kernel-grade	kernel_score $\geq$ 60; os_score > 40	Contains substantial core primitives and enough platform code to act as a standalone or research kernel.
Kernel-prototype	30 $\leq$ kernel_score < 60	Implements some core primitives but is missing critical parts; may be a research experiment or early prototype.
OS-platform	kernel_score < 30 and os_score $\geq$ 50	Lacks kernel primitives but provides platform services (e.g., drivers, filesystems) on top of an existing kernel.
Application / other	kernel_score < 30 and os_score < 50	Primarily user-space code or unrelated library; not a kernel or OS.

These thresholds can be adjusted to suit specific evaluation goals. For AGI-OS readiness, a repository should aim for kernel_score $\geq$ 70 and os_score $\geq$ 70.

5 How to apply this rubric

1. **Extract manifest and status data.** Use the Echo.Kern function manifest and status report as the canonical list of required functions. For each category, note the target functions/SLOC [1]. Update this list as the kernel evolves.
2. **Analyse the repository.** Use static analysis tools or manual inspection to identify files and functions that match each category. Count matching functions or SLOC to estimate completeness. Searching for keywords and comparing function prototypes is a practical starting point.
3. **Compute per-category scores.** Determine presence (0, 0.5, 1) and completeness ratio, multiply by the weight and sum across categories. Normalise to derive kernel and OS scores.
4. **Interpret results.** Use the classification table to determine whether the repository behaves like a kernel, an OS platform, an application, or something else. Document which features are present and which are missing to guide integration efforts.

6 Example

Suppose repository **X** contains a `scheduler.c` with three scheduling functions, a `memory.c` with a simple allocator, and a `syscalls.c` exposing five basic system calls. It lacks boot code and interrupt handlers. Presence scores for scheduling, memory management and system calls are 1 (evidence exists); completeness for those categories might be 3/18 = 0.17, 1/24 = 0.04 and 5/32 = 0.16 respectively. Scores would be:

```
Boot = 10 × 0 × 0 = 0
Scheduling = 9 × 1 × 0.17 ≈ 1.53
Process mgmt = 8 × 0 × 0 = 0 (absent)
Memory mgmt = 8 × 1 × 0.04 ≈ 0.32
Interrupts = 6 × 0 × 0 = 0
Syscalls = 5 × 1 × 0.16 ≈ 0.80
I/O = 5 × 0 × 0 = 0
Synchronisation = 4 × 0 × 0 = 0
Timers = 3 × 0 × 0 = 0
Protection = 2 × 0 × 0 = 0
Total core score = 1.53 + 0.32 + 0.80 = 2.65
kernel_score = 2.65 / 60 × 100 ≈ 4.4 (very low)
```

The OS-level categories would likely also score near zero. **X** would therefore classify as *application/other* (contains fragments of kernel logic but not enough to be considered a kernel).

7 Extending the rubric

The rubric can be refined by:

- **Adding sub-weights within categories** based on the manifest's priority (critical/high/medium/low). For example, within the scheduler category, context switching functions may receive higher weight than secondary scheduling policies. This allows more granular scoring.
- **Incorporating runtime tests.** Beyond static analysis, use automated tests to validate functional behaviour (e.g., does the scheduler enforce time-slicing? does memory allocation protect against overlaps?). Pass/fail outcomes could adjust completeness scores.

- **Customising for domains.** If evaluating research kernels for neuromorphic computing, emphasise ESN reservoir and P-system primitives (from the Echo.Kern manifest) or include extension categories from the manifest's "Extensions" section, adjusting the weights accordingly.

8 Conclusion

By anchoring the evaluation to a clear taxonomy of kernel and OS primitives [1] and using weighted scores to reflect presence and completeness, this rubric provides a comprehensive and quantitative method to assess repositories for their suitability as AGI-OS kernels or platforms. The rubric is flexible and can evolve as new primitives emerge, ensuring that evaluations remain aligned with the state-of-the-art.

---

[1] Kernel (operating system) - Wikipedia
https://en.wikipedia.org/wiki/Kernel_(operating_system)