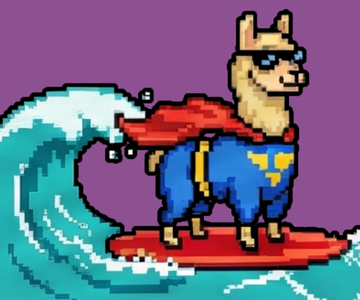
INSTALLAMA.SH

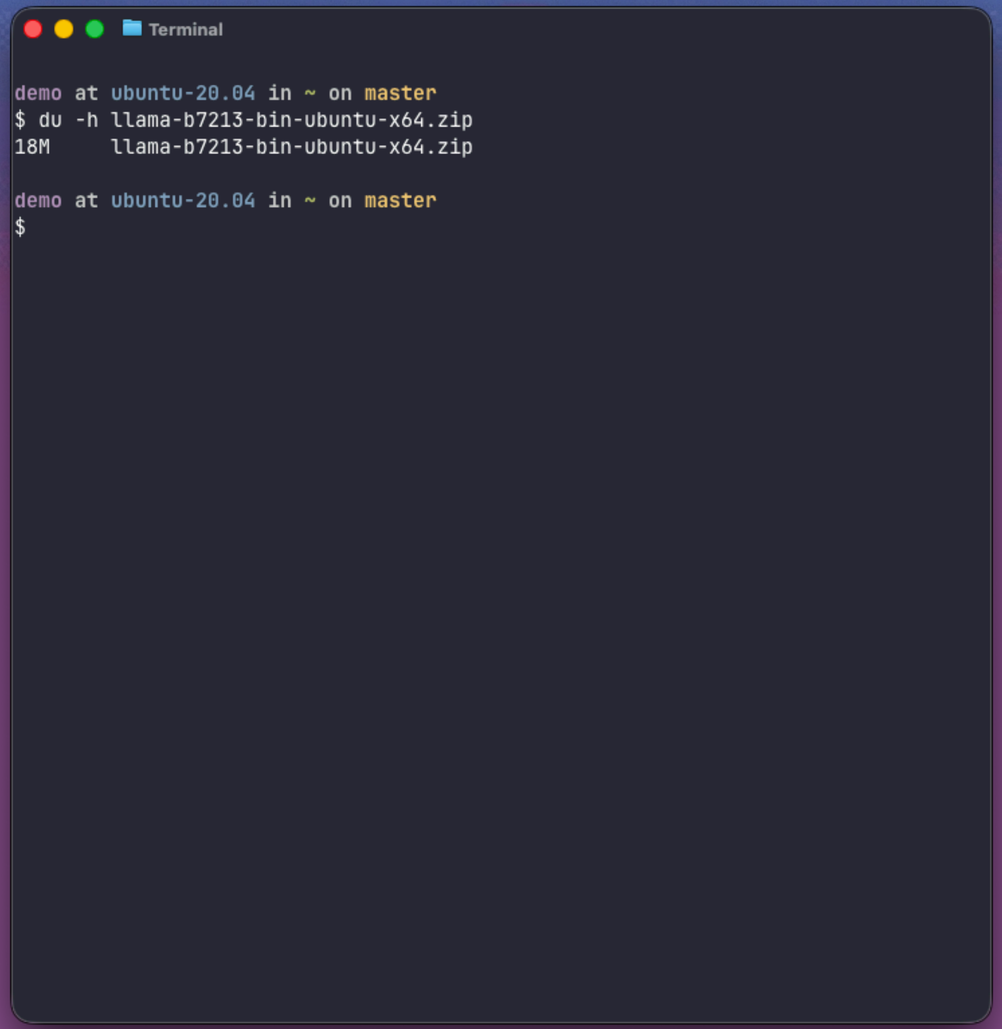
ANGT - HUGGING FACE - 2025

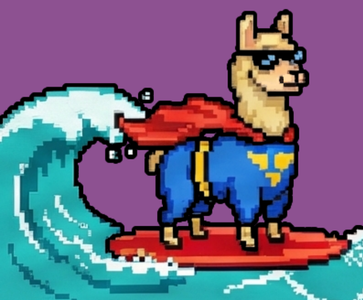


THE DISTRIBUTION DILEMMA

Performance is critical for llama.cpp, yet distributing it remains a challenge.

Let's look at the current status for Linux:



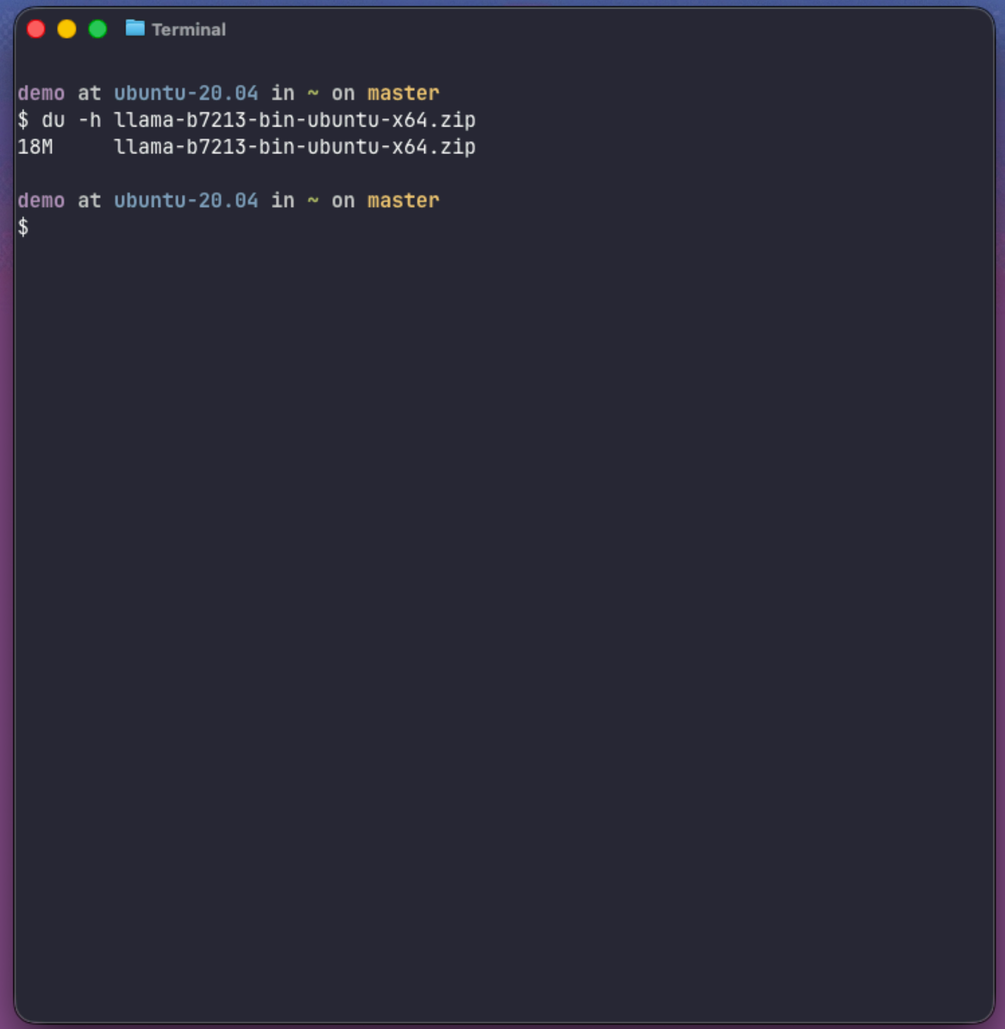


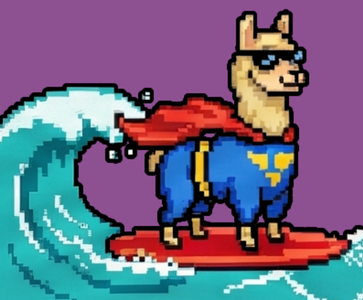
THE DISTRIBUTION DILEMMA

Performance is critical for llama.cpp, yet distributing it remains a challenge.

Let's look at the current status for Linux:

- Only Ubuntu is officially supported



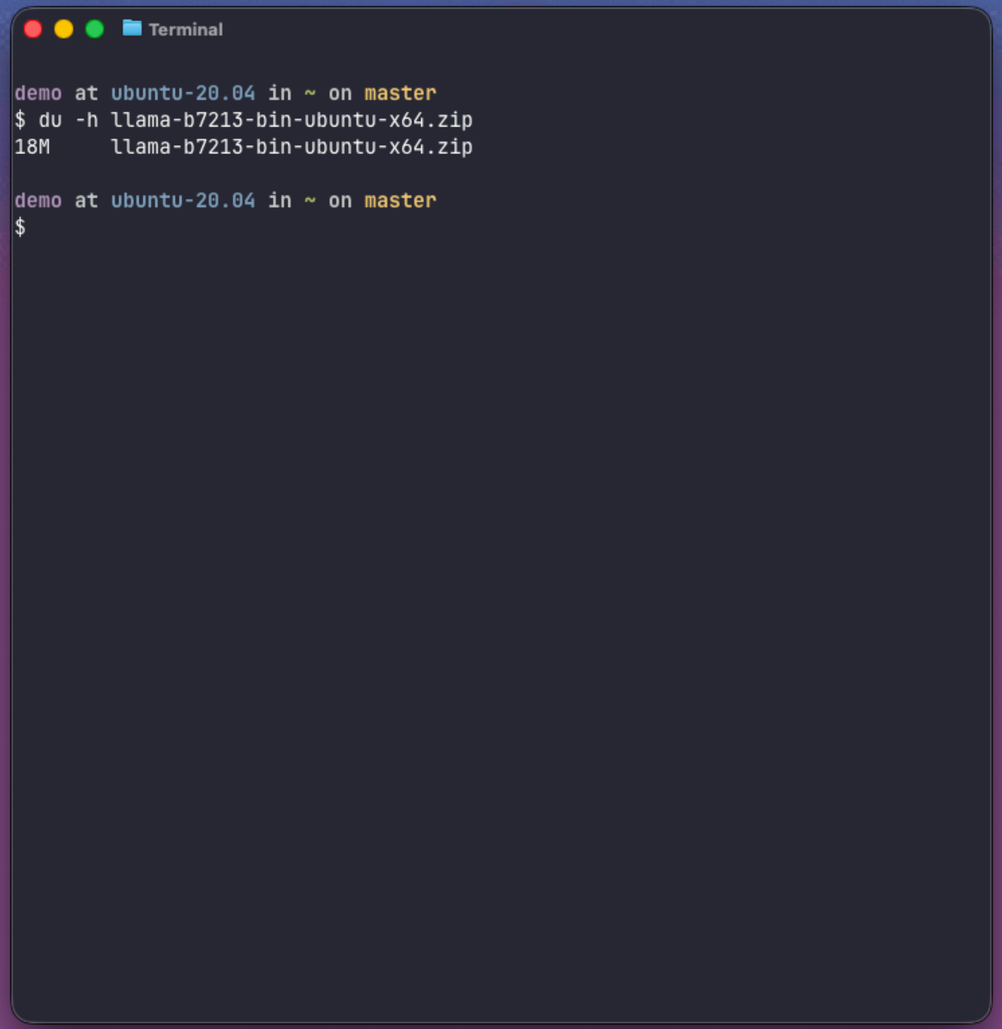


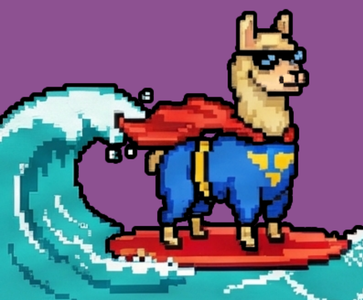
THE DISTRIBUTION DILEMMA

Performance is critical for llama.cpp, yet distributing it remains a challenge.

Let's look at the current status for Linux:

- Only Ubuntu is officially supported
- No binaries for Linux ARM64



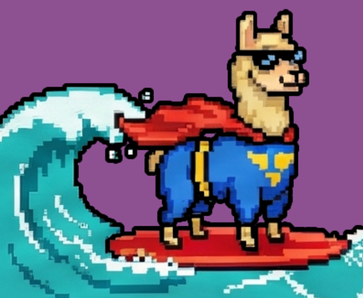


THE DISTRIBUTION DILEMMA

Performance is critical for llama.cpp, yet distributing it remains a challenge.

Let's look at the current status for Linux:

- Only Ubuntu is officially supported
- No binaries for Linux ARM64
- Limited CPU compatibility



Terminal

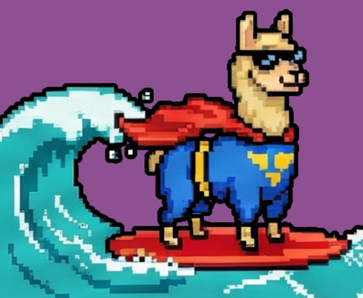
```
demo at ubuntu-20.04 in ~ on master
$ du -h llama-b7213-bin-ubuntu-x64.zip
18M     llama-b7213-bin-ubuntu-x64.zip

demo at ubuntu-20.04 in ~ on master
$ unzip llama-b7213-bin-ubuntu-x64.zip
Archive:  llama-b7213-bin-ubuntu-x64.zip
  inflating: build/bin/LICENSE
  inflating: build/bin/LICENSE-curl
  inflating: build/bin/LICENSE-httplib
  inflating: build/bin/LICENSE-jsonhpp
  inflating: build/bin/LICENSE-linenoise
  inflating: build/bin/libggml-base.so
  inflating: build/bin/libggml-base.so.0
  inflating: build/bin/libggml-base.so.0.9.4
  inflating: build/bin/libggml-cpu-alderlake.so
  inflating: build/bin/libggml-cpu-haswell.so
  inflating: build/bin/libggml-cpu-icelake.so
  inflating: build/bin/libggml-cpu-sandybridge.so
  inflating: build/bin/libggml-cpu-sapphirerapids.so
  inflating: build/bin/libggml-cpu-skylakex.so
  inflating: build/bin/libggml-cpu-sse42.so
  inflating: build/bin/libggml-cpu-x64.so
  inflating: build/bin/libggml-rpc.so
  inflating: build/bin/libggml.so
  inflating: build/bin/libggml.so.0
  inflating: build/bin/libggml.so.0.9.4
  inflating: build/bin/libllama.so
  inflating: build/bin/libllama.so.0
  inflating: build/bin/libllama.so.0.0.7213
  inflating: build/bin/libmtmd.so
  inflating: build/bin/libmtmd.so.0
  inflating: build/bin/libmtmd.so.0.0.7213
  inflating: build/bin/llama-batched-bench
  inflating: build/bin/llama-bench
```

THE DISTRIBUTION DILEMMA

Performance is critical for llama.cpp, yet distributing it remains a challenge.

Let's look at the current status for Linux:

- Only Ubuntu is officially supported
- No binaries for Linux ARM64
- Limited CPU compatibility
- Incompatible with older distributions



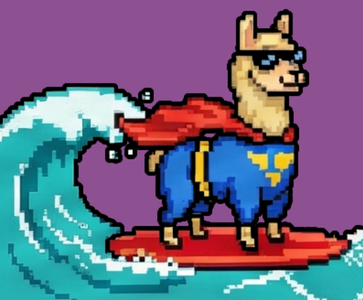
Terminal

```
  inflating: build/bin/llama-tts
  inflating: build/bin/rpc-server

demo at ubuntu-20.04 in ~ on master
$ build/bin/llama-server
build/bin/llama-server: /lib/x86_64-linux-gnu/libc.so.6: version `GLIBC_2.32' no
t found (required by build/bin/llama-server)
build/bin/llama-server: /lib/x86_64-linux-gnu/libc.so.6: version `GLIBC_2.34' no
t found (required by build/bin/llama-server)
build/bin/llama-server: /lib/x86_64-linux-gnu/libc.so.6: version `GLIBC_2.33' no
t found (required by build/bin/llama-server)
build/bin/llama-server: /lib/x86_64-linux-gnu/libstdc++.so.6: version `GLIBCXX_3
.4.30' not found (required by build/bin/llama-server)
build/bin/llama-server: /lib/x86_64-linux-gnu/libstdc++.so.6: version `GLIBCXX_3
.4.29' not found (required by build/bin/llama-server)
build/bin/llama-server: /lib/x86_64-linux-gnu/libstdc++.so.6: version `CXXABI_1.
3.13' not found (required by build/bin/llama-server)
build/bin/llama-server: /lib/x86_64-linux-gnu/libstdc++.so.6: version `GLIBCXX_3
.4.29' not found (required by /home/demo/build/bin/libmtmd.so.0)
build/bin/llama-server: /lib/x86_64-linux-gnu/libstdc++.so.6: version `GLIBCXX_3
.4.29' not found (required by /home/demo/build/bin/libllama.so.0)
build/bin/llama-server: /lib/x86_64-linux-gnu/libstdc++.so.6: version `CXXABI_1.
3.13' not found (required by /home/demo/build/bin/libllama.so.0)
build/bin/llama-server: /lib/x86_64-linux-gnu/libc.so.6: version `GLIBC_2.32' no
t found (required by /home/demo/build/bin/libllama.so.0)
build/bin/llama-server: /lib/x86_64-linux-gnu/libc.so.6: version `GLIBC_2.34' no
t found (required by /home/demo/build/bin/libllama.so.0)
build/bin/llama-server: /lib/x86_64-linux-gnu/libc.so.6: version `GLIBC_2.32' no
t found (required by /home/demo/build/bin/libggml.so.0)
build/bin/llama-server: /lib/x86_64-linux-gnu/libc.so.6: version `GLIBC_2.34' no
t found (required by /home/demo/build/bin/libggml.so.0)
build/bin/llama-server: /lib/x86_64-linux-gnu/libstdc++.so.6: version `GLIBCXX_3
.4.29' not found (required by /home/demo/build/bin/libggml-base.so.0)

demo at ubuntu-20.04 in ~ on master
$
```

THE DISTRIBUTION DILEMMA

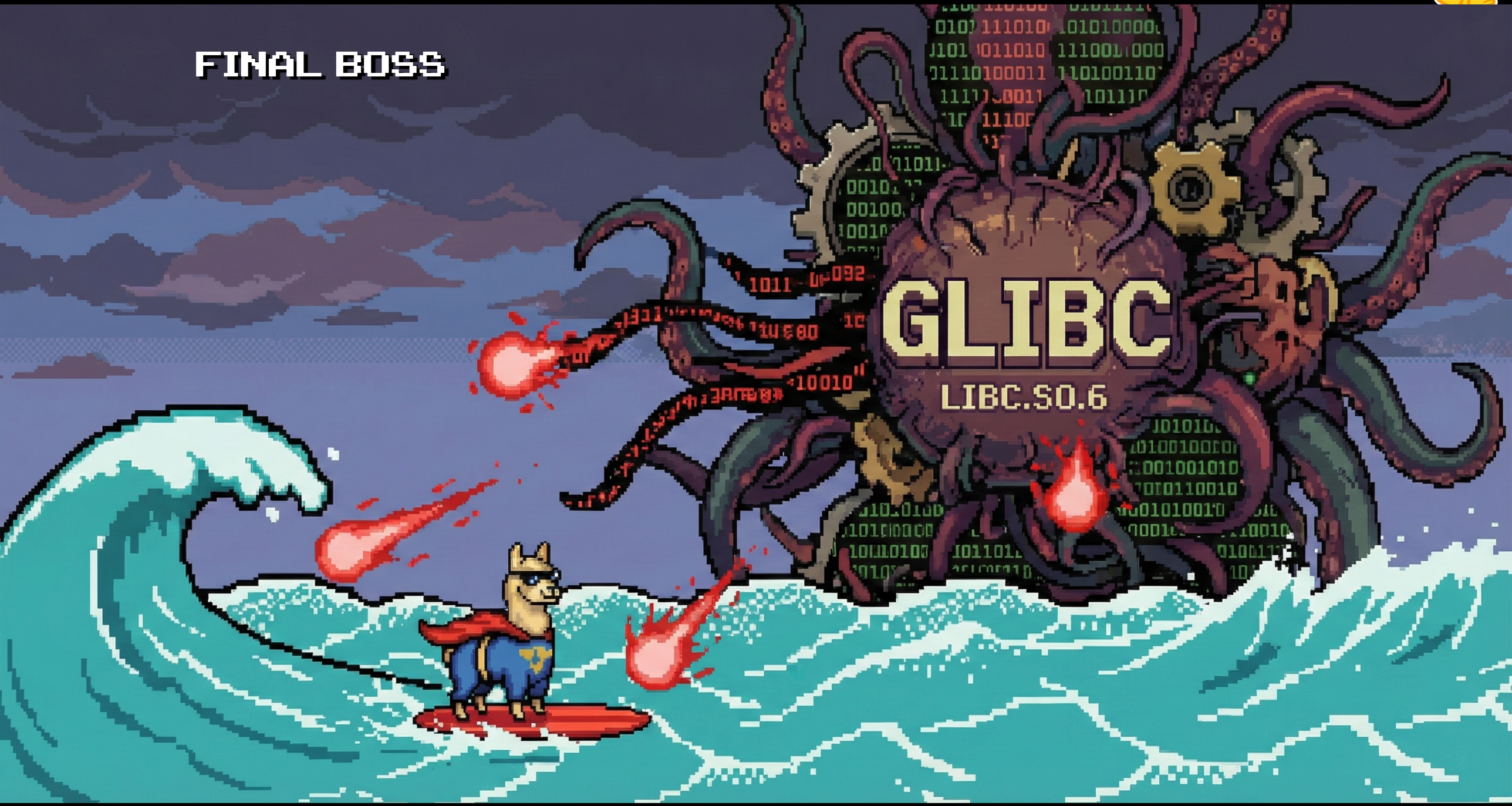
Dynamic linking relies on the host system.

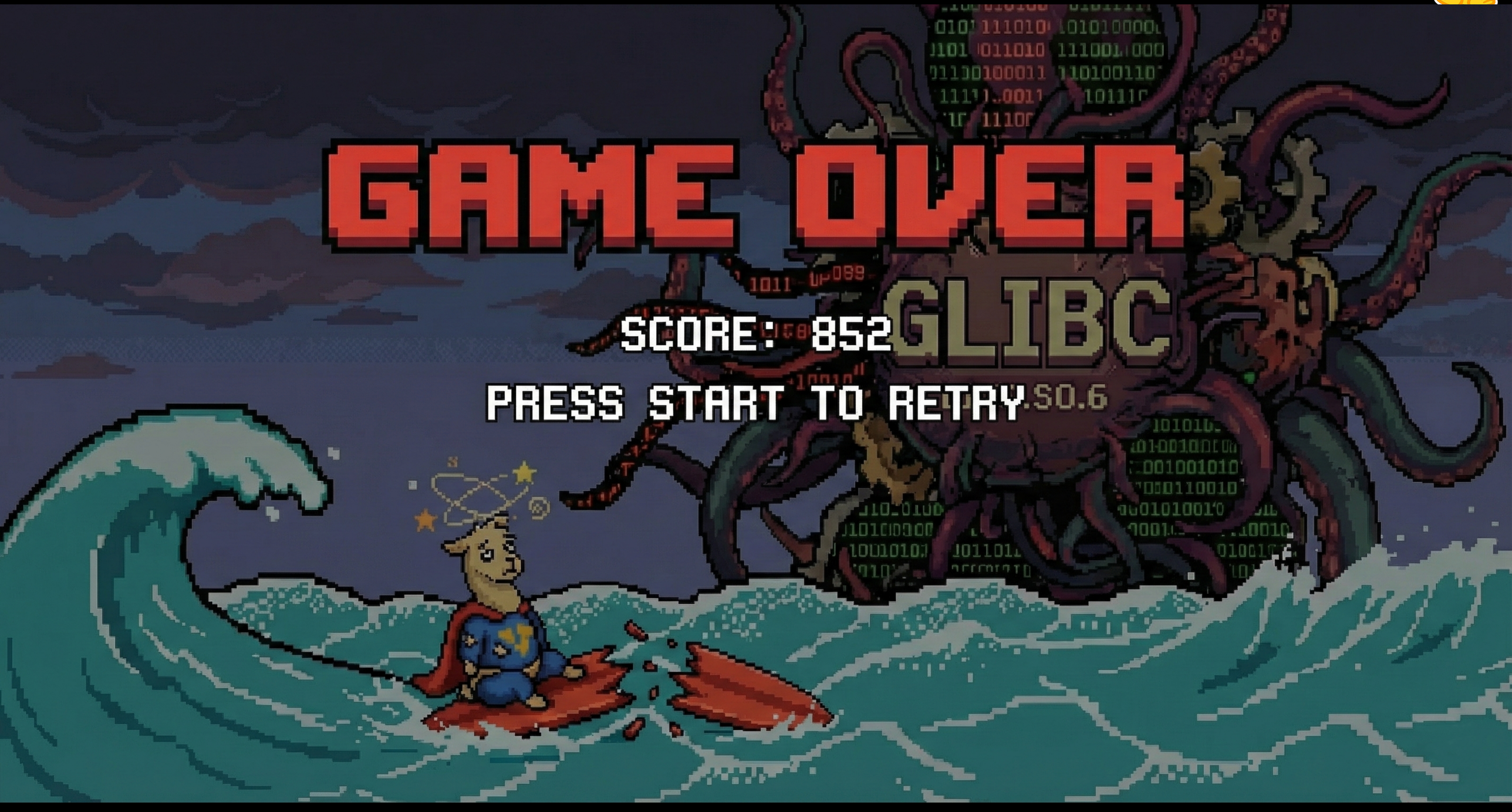
Sadly, Linux has no universal standard.



Terminal

```
  inflating: build/bin/llama-tts
  inflating: build/bin/rpc-server

demo at ubuntu-20.04 in ~ on master
$ build/bin/llama-server
build/bin/llama-server: /lib/x86_64-linux-gnu/libc.so.6: version `GLIBC_2.32' no
t found (required by build/bin/llama-server)
build/bin/llama-server: /lib/x86_64-linux-gnu/libc.so.6: version `GLIBC_2.34' no
t found (required by build/bin/llama-server)
build/bin/llama-server: /lib/x86_64-linux-gnu/libc.so.6: version `GLIBC_2.33' no
t found (required by build/bin/llama-server)
build/bin/llama-server: /lib/x86_64-linux-gnu/libstdc++.so.6: version `GLIBCXX_3
.4.30' not found (required by build/bin/llama-server)
build/bin/llama-server: /lib/x86_64-linux-gnu/libstdc++.so.6: version `GLIBCXX_3
.4.29' not found (required by build/bin/llama-server)
build/bin/llama-server: /lib/x86_64-linux-gnu/libstdc++.so.6: version `CXXABI_1.
3.13' not found (required by build/bin/llama-server)
build/bin/llama-server: /lib/x86_64-linux-gnu/libstdc++.so.6: version `GLIBCXX_3
.4.29' not found (required by /home/demo/build/bin/libmtmd.so.0)
build/bin/llama-server: /lib/x86_64-linux-gnu/libstdc++.so.6: version `GLIBCXX_3
.4.29' not found (required by /home/demo/build/bin/libllama.so.0)
build/bin/llama-server: /lib/x86_64-linux-gnu/libstdc++.so.6: version `CXXABI_1.
3.13' not found (required by /home/demo/build/bin/libllama.so.0)
build/bin/llama-server: /lib/x86_64-linux-gnu/libc.so.6: version `GLIBC_2.32' no
t found (required by /home/demo/build/bin/libllama.so.0)
build/bin/llama-server: /lib/x86_64-linux-gnu/libc.so.6: version `GLIBC_2.34' no
t found (required by /home/demo/build/bin/libllama.so.0)
build/bin/llama-server: /lib/x86_64-linux-gnu/libc.so.6: version `GLIBC_2.32' no
t found (required by /home/demo/build/bin/libggml.so.0)
build/bin/llama-server: /lib/x86_64-linux-gnu/libc.so.6: version `GLIBC_2.34' no
t found (required by /home/demo/build/bin/libggml.so.0)
build/bin/llama-server: /lib/x86_64-linux-gnu/libstdc++.so.6: version `GLIBCXX_3
.4.29' not found (required by /home/demo/build/bin/libggml-base.so.0)

demo at ubuntu-20.04 in ~ on master
$
```

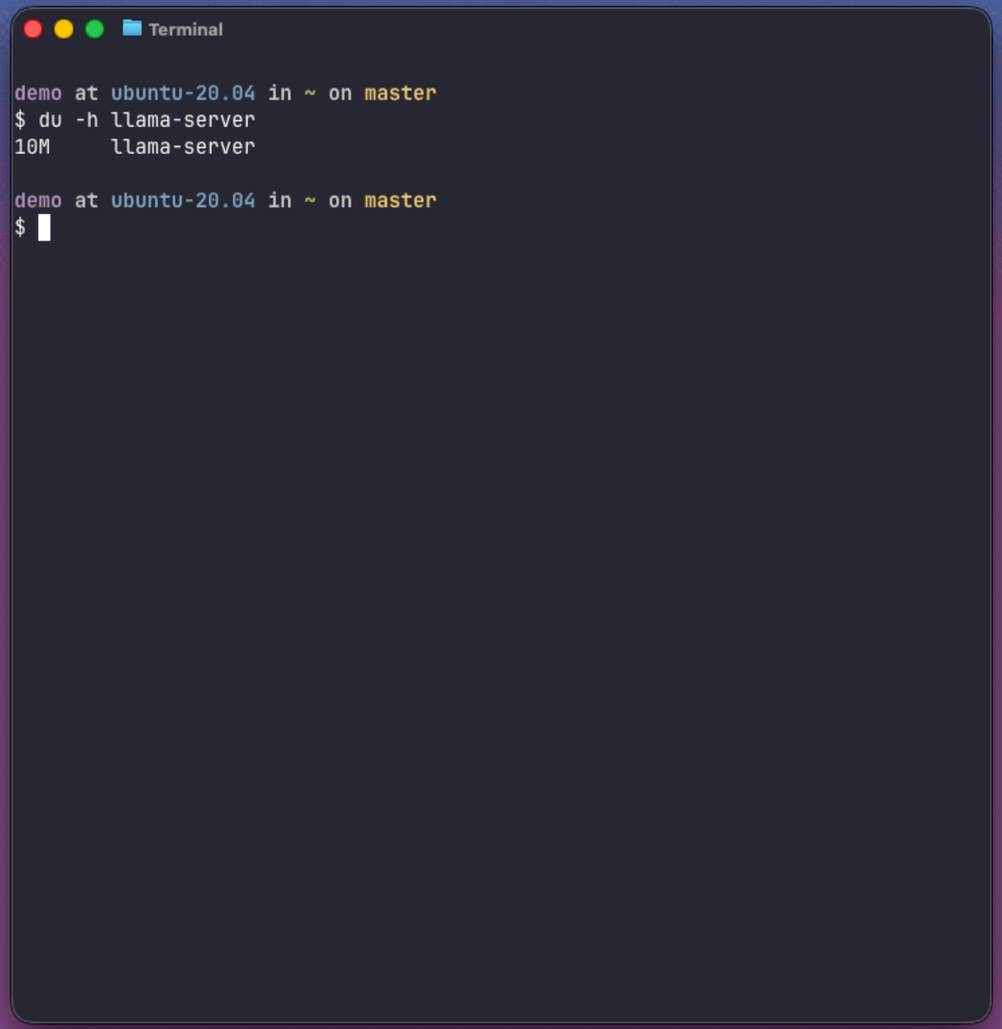

FINAL BOSS
GLIBC
LIBC.SO.6

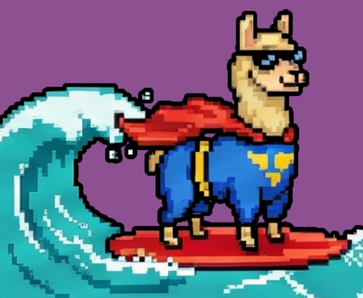

GAME OVER
SCORE: 852
PRESS START TO RETRY
GLIBC

PRESS START

Let’s restart the game!

We can compile llama.cpp without the dynamic loader that tries to detect all the features at runtime:



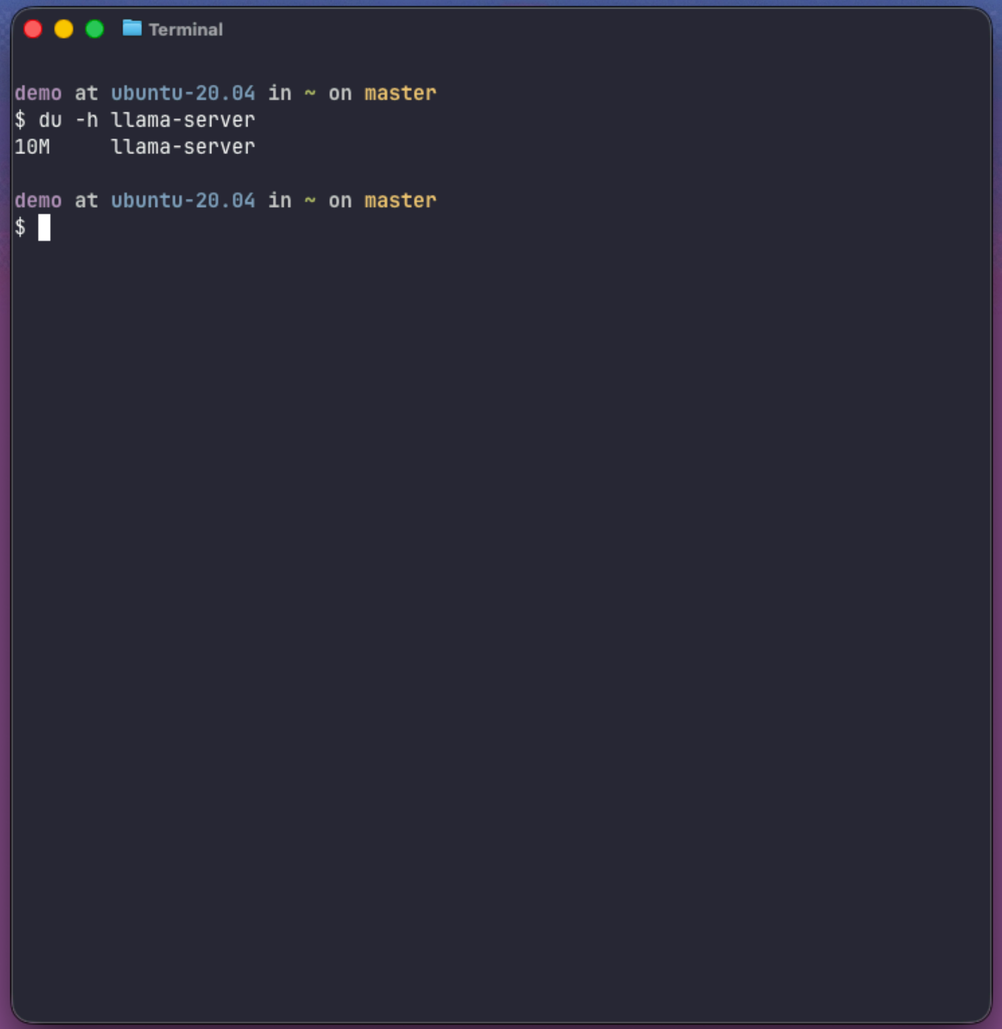


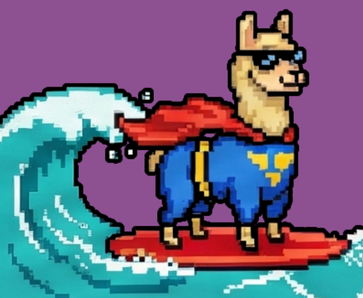
PRESS START

Let’s restart the game!

We can compile llama.cpp without the dynamic loader that tries to detect all the features at runtime:

- Optimized size: a smaller binary to download





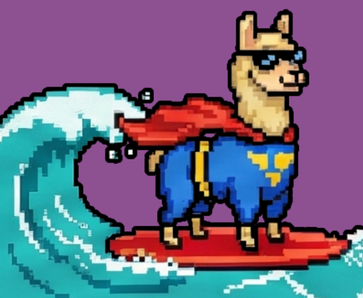
PRESS START

Let’s restart the game!

We can compile llama.cpp without the dynamic loader that tries to detect all the features at runtime:

- Optimized size: a smaller binary to download
- Truly portable: a statically linked binary that works on any Linux distribution

```
Terminal

demo at ubuntu-20.04 in ~ on master
$ du -h llama-server
10M     llama-server

demo at ubuntu-20.04 in ~ on master
$ ldd llama-server
        not a dynamic executable

demo at ubuntu-20.04 in ~ on master
$
```



PRESS START

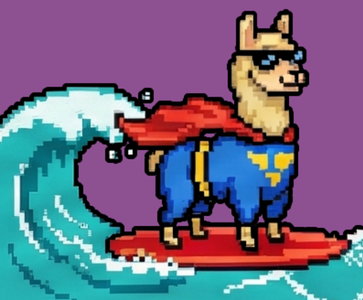
Let’s restart the game!

We can compile llama.cpp without the dynamic loader that tries to detect all the features at runtime:

- Optimized size: a smaller binary to download
- Truly portable: a statically linked binary that works on any Linux distribution
- Zero overhead: no runtime dispatch

Terminal

```
demo at ubuntu-20.04 in ~ on master
$ du -h llama-server
10M     llama-server

demo at ubuntu-20.04 in ~ on master
$ ldd llama-server
        not a dynamic executable

demo at ubuntu-20.04 in ~ on master
$ ./llama-server
main: setting n_parallel = 4 and kv_unified = true (add -kvu to disable this)
build: 7234 (84ecf289) with Clang 20.1.2 for Linux x86_64
system info: n_threads = 8, n_threads_batch = 8, total_threads = 8

system_info: n_threads = 8 (n_threads_batch = 8) / 8 | CPU : SSE3 = 1 | SSSE3 =
1 | AVX = 1 | AVX2 = 1 | F16C = 1 | FMA = 1 | BMI2 = 1 | LLAMAFILE = 1 | REPACK
= 1 |

Running without SSL
init: using 7 threads for HTTP server
main: starting router server, no model will be loaded in this process
start: binding port with default address family
main: router server is listening on http://127.0.0.1:8080
main: NOTE: router mode is experimental
main:       it is not recommended to use this mode in untrusted environments
```



THE SECRET WEAPON

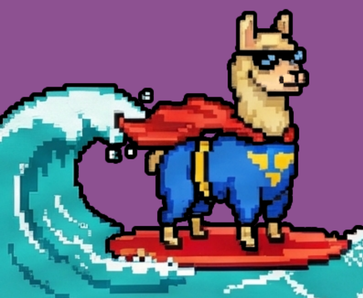
There is a robust solution: just run the optimized code! If the CPU doesn't support it, it emits a SIGILL, and, we can trap it!

The plan is simple:

Terminal

```
#else
#define FEATURE(feat, name) [feat]={name,NULL}
#endif

enum {
    avx, f16c, fma, avx2, bmi2,
    avxvnni, avxvnniint8,
    avx512f, avx512vl, avx512bw, avx512cd, avx512dq,
    avx512vnni, avx512vbmi, avx512bf16,
    amxtile, amxint8, amxbf16
};

static struct feature
x86_64_list[] = {
    FEATURE(avx,         "avx"        ),
    FEATURE(f16c,        "f16c"       ),
    FEATURE(fma,         "fma"        ),
    FEATURE(avx2,        "avx2"       ),
    FEATURE(bmi2,        "bmi2"       ),
    FEATURE(avxvnni,     "avxvnni"    ),
    FEATURE(avxvnniint8, "avxvnniint8"),
    FEATURE(avx512f,     "avx512f"    ),
    FEATURE(avx512vl,    "avx512vl"   ),
    FEATURE(avx512bw,    "avx512bw"   ),
    FEATURE(avx512dq,    "avx512dq"   ),
    FEATURE(avx512cd,    "avx512cd"   ),
    FEATURE(avx512vnni,  "avx512vnni" ),
    FEATURE(avx512vbmi,  "avx512vbmi" ),
    FEATURE(avx512bf16,  "avx512bf16" ),
    FEATURE(amxtile,     "amx-tile"   ),
    FEATURE(amxint8,     "amx-int8"   ),
    FEATURE(amxbf16,     "amx-bf16"   ),
};
```

156,1 82%



THE SECRET WEAPON

There is a robust solution: just run the optimized code! If the CPU doesn't support it, it emits a SIGILL, and, we can trap it!

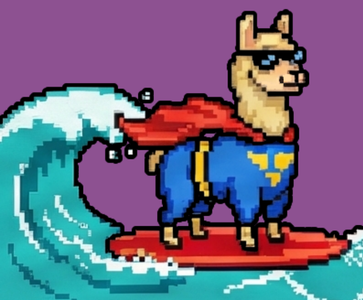
The plan is simple:

- List all features llama.cpp can uses

Terminal

```
#else
#define FEATURE(feat, name) [feat]={name,NULL}
#endif

enum {
    avx, f16c, fma, avx2, bmi2,
    avxvnni, avxvnniint8,
    avx512f, avx512vl, avx512bw, avx512cd, avx512dq,
    avx512vnni, avx512vbmi, avx512bf16,
    amxtile, amxint8, amxbf16
};

static struct feature
x86_64_list[] = {
    FEATURE(avx,         "avx"        ),
    FEATURE(f16c,        "f16c"       ),
    FEATURE(fma,         "fma"        ),
    FEATURE(avx2,        "avx2"       ),
    FEATURE(bmi2,        "bmi2"       ),
    FEATURE(avxvnni,     "avxvnni"    ),
    FEATURE(avxvnniint8, "avxvnniint8"),
    FEATURE(avx512f,     "avx512f"    ),
    FEATURE(avx512vl,    "avx512vl"   ),
    FEATURE(avx512bw,    "avx512bw"   ),
    FEATURE(avx512dq,    "avx512dq"   ),
    FEATURE(avx512cd,    "avx512cd"   ),
    FEATURE(avx512vnni,  "avx512vnni" ),
    FEATURE(avx512vbmi,  "avx512vbmi" ),
    FEATURE(avx512bf16,  "avx512bf16" ),
    FEATURE(amxtile,     "amx-tile"   ),
    FEATURE(amxint8,     "amx-int8"   ),
    FEATURE(amxbf16,     "amx-bf16"   ),
};
```

156,1 82%



THE SECRET WEAPON

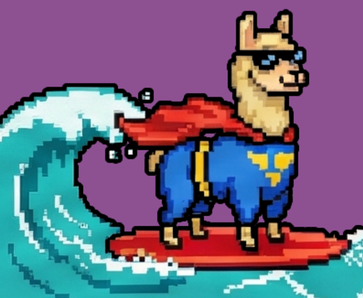
There is a robust solution: just run the optimized code! If the CPU doesn't support it, it emits a SIGILL, and, we can trap it!

The plan is simple:

- List all features llama.cpp can uses
- Execute a tiny snippet of raw machine code for each

Terminal

```
static void __attribute__((target("fma")))
check_fma(void) {
    asm volatile("vfmadd132ps %%ymm0, %%ymm1, %%ymm2" ::: "ymm0", "ymm1", "ymm2"
}

static void __attribute__((target("avx2")))
check_avx2(void) {
    asm volatile("vpaddd %%ymm0, %%ymm1, %%ymm2" ::: "ymm0", "ymm1", "ymm2");
}

static void __attribute__((target("bmi2")))
check_bmi2(void) {
    asm volatile("pext %%rax, %%rbx, %%rcx" ::: "rax", "rbx", "rcx");
}

static void __attribute__((target("avxvnni")))
check_avxvnni(void) { // vpdpwssd
    asm volatile(".byte 0xc4, 0xe2, 0x75, 0x52, 0xd0" ::: "ymm0", "ymm1", "ymm2"
}

static void __attribute__((target("avxvnniint8")))
check_avxvnniint8(void) {
    asm volatile("vpdpbssd %%ymm0, %%ymm1, %%ymm2" ::: "ymm0", "ymm1", "ymm2");
}

static void __attribute__((target("evex512,avx512f")))
check_avx512f(void) {
    asm volatile("vaddps %%zmm0, %%zmm1, %%zmm2" ::: "zmm0", "zmm1", "zmm2");
}

static void __attribute__((target("evex512,avx512vl")))
check_avx512vl(void) {
    asm volatile("vpternlogd $0xFF, %%ymm0, %%ymm1, %%ymm2" ::: "ymm0", "ymm1",
}
                                                    32,1          8%
```



THE SECRET WEAPON

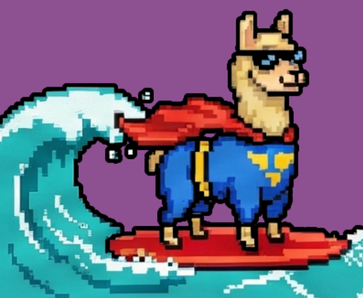
There is a robust solution: just run the optimized code! If the CPU doesn't support it, it emits a SIGILL, and, we can trap it!

The plan is simple:

- List all features llama.cpp can uses
- Execute a tiny snippet of raw machine code for each
- Trap the signal to verify if it worked or not

Terminal

```
static void
detect(struct features features)
{
    struct sigaction sa = {
        .sa_handler = handler
    };
    sigaction(SIGILL, &sa, NULL);

    for (size_t i = 0; i < features.count; i++) {
        if (features.list[i].detect && !sigsetjmp(jmp, 1)) {
            features.list[i].detect();
            features.list[i].set = 1;
        }
    }
}

static void
decode(struct features features, const char *code)
{
    for (size_t i = 0; i < features.count; i++) {
        char c = code[i / 4];

        if (!c)
            break;

        if (((c - 'k') >> (i & 3)) & 1)
            features.list[i].set = 1;
    }
}

static int
encode(struct features features)
{
    int h = 0;
                                                    87,0-1        79%
```



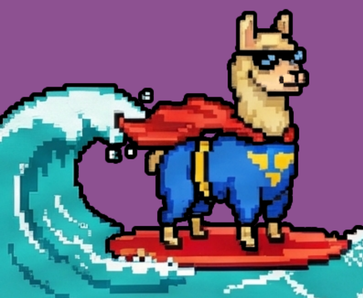
THE SECRET WEAPON

Bonus point:

The exact same technic works for ARM64!

Terminal

```
static void __attribute__((target("+sve")))
check_sve(void) {
    asm volatile("incw x0" ::: "x0");
}

static void __attribute__((target("+sve2")))
check_sve2(void) {
    asm volatile("smlalb z0.s, z0.h, z0.h" ::: "z0");
}

static void __attribute__((target("+sme")))
check_sme(void) {
    asm volatile("smstart sm\n\tsmstop sm");
}

#else
#define FEATURE(feat, name) [feat]={name,NULL}
#endif

enum {
    fp16,
    dotprod, i8mm,
    sve, sve2, sme
};

static struct feature
aarch64_list[] = {
    FEATURE(fp16,    "fp16"   ),
    FEATURE(dotprod, "dotprod"),
    FEATURE(i8mm,    "i8mm"   ),
    FEATURE(sve,     "sve"    ),
    FEATURE(sve2,    "sve2"   ),
    FEATURE(sme,     "sme"    ),
};
                                                    37,1          54%
```



THE LAST WALL

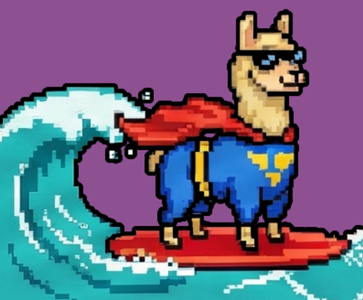
For x86_64 alone, we have 18 features:

2^18 = 262,144 combinations.

But we don’t have to build all of them if we find all their dependencies:

Terminal

```
# strict
('f16c',         'avx'       ),
('fma',          'avx'       ),
('avx2',         'avx'       ),
('avxvnni',      'avx2'      ),
('avxvnniint8',  'avx2'      ),
('avx512f',      'avx2'      ),
('avx512f',      'f16c'      ),
('avx512f',      'fma'       ),
('avx512vl',     'avx512f'   ),
('avx512bw',     'avx512f'   ),
('avx512dq',     'avx512f'   ),
('avx512cd',     'avx512f'   ),
('avx512vnni',   'avx512f'   ),
('avx512vbmi',   'avx512bw'  ),
('avx512bf16',   'avx512bw'  ),
('amx-int8',     'amx-tile'  ),
('amx-bf16',     'amx-tile'  ),
# observed so far
('fma',          'f16c'      ),
('avx2',         'bmi2'      ),
('avx2',         'fma'       ),
('bmi2',         'avx2'      ),
('avxvnniint8',  'avxvnni'   ),
('avx512f',      'avx512cd'  ),
('avx512bw',     'avx512dq'  ),
('avx512dq',     'avx512vl'  ),
('avx512vl',     'avx512bw'  ),
('avx512vnni',   'avx512bw'  ),
('amx-tile',     'avxvnni'   ),
('amx-tile',     'avx512vnni'),
('amx-tile',     'avx512vbmi'),
('amx-tile',     'avx512bf16'),
('amx-tile',     'amx-int8'  ),
('amx-tile',     'amx-bf16'  ),
                                94,1          15%
```



THE LAST WALL

For x86_64 alone, we have 18 features:

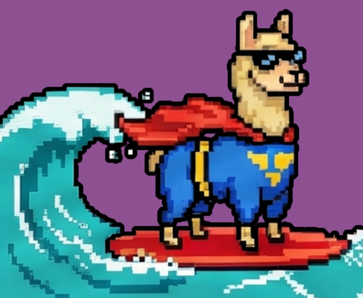
2^18 = 262,144 combinations.

But we don’t have to build all of them if we find all their dependencies:

- Strict: for example, AVX is a subset of AVX2 by definition… These rules drop the count to ~20k

Terminal

```
# strict
('f16c',         'avx'       ),
('fma',          'avx'       ),
('avx2',         'avx'       ),
('avxvnni',      'avx2'      ),
('avxvnniint8',  'avx2'      ),
('avx512f',      'avx2'      ),
('avx512f',      'f16c'      ),
('avx512f',      'fma'       ),
('avx512vl',     'avx512f'   ),
('avx512bw',     'avx512f'   ),
('avx512dq',     'avx512f'   ),
('avx512cd',     'avx512f'   ),
('avx512vnni',   'avx512f'   ),
('avx512vbmi',   'avx512bw'  ),
('avx512bf16',   'avx512bw'  ),
('amx-int8',     'amx-tile'  ),
('amx-bf16',     'amx-tile'  ),
# observed so far
('fma',          'f16c'      ),
('avx2',         'bmi2'      ),
('avx2',         'fma'       ),
('bmi2',         'avx2'      ),
('avxvnniint8',  'avxvnni'   ),
('avx512f',      'avx512cd'  ),
('avx512bw',     'avx512dq'  ),
('avx512dq',     'avx512vl'  ),
('avx512vl',     'avx512bw'  ),
('avx512vnni',   'avx512bw'  ),
('amx-tile',     'avxvnni'   ),
('amx-tile',     'avx512vnni'),
('amx-tile',     'avx512vbmi'),
('amx-tile',     'avx512bf16'),
('amx-tile',     'amx-int8'  ),
('amx-tile',     'amx-bf16'  ),
                                94,1          15%
```



THE LAST WALL

For x86_64 alone, we have 18 features:

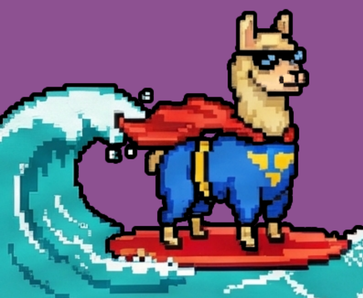
2^18 = 262,144 combinations.

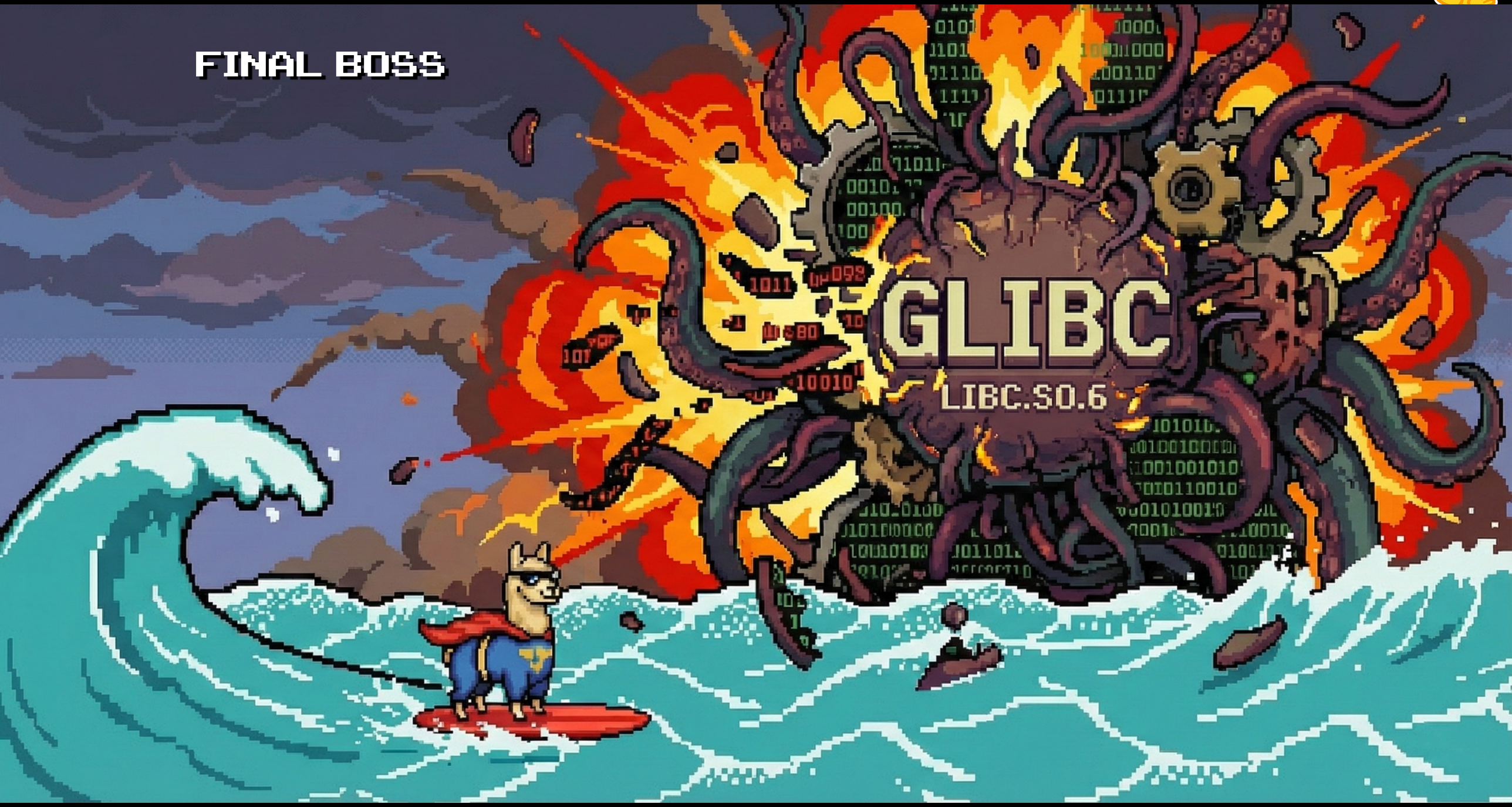
But we don't have to build all of them if we find all their dependencies:

- Strict: for example, AVX is a subset of AVX2 by definition… These rules drop the count to ~20k
- Observed: after a meticulous exploration of all released CPUs, the count drops to 36

Terminal

```
# strict
('f16c',         'avx'       ),
('fma',          'avx'       ),
('avx2',         'avx'       ),
('avxvnni',      'avx2'      ),
('avxvnniint8',  'avx2'      ),
('avx512f',      'avx2'      ),
('avx512f',      'f16c'      ),
('avx512f',      'fma'       ),
('avx512vl',     'avx512f'   ),
('avx512bw',     'avx512f'   ),
('avx512dq',     'avx512f'   ),
('avx512cd',     'avx512f'   ),
('avx512vnni',   'avx512f'   ),
('avx512vbmi',   'avx512bw'  ),
('avx512bf16',   'avx512bw'  ),
('amx-int8',     'amx-tile'  ),
('amx-bf16',     'amx-tile'  ),
# observed so far
('fma',          'f16c'      ),
('avx2',         'bmi2'      ),
('avx2',         'fma'       ),
('bmi2',         'avx2'      ),
('avxvnniint8',  'avxvnni'   ),
('avx512f',      'avx512cd'  ),
('avx512bw',     'avx512dq'  ),
('avx512dq',     'avx512vl'  ),
('avx512vl',     'avx512bw'  ),
('avx512vnni',   'avx512bw'  ),
('amx-tile',     'avxvnni'   ),
('amx-tile',     'avx512vnni'),
('amx-tile',     'avx512vbmi'),
('amx-tile',     'avx512bf16'),
('amx-tile',     'amx-int8'  ),
('amx-tile',     'amx-bf16'  ),
                                94,1          15%
```




FINAL BOSS
GLIBC
LIBC.SO.6

THE END