HarfBuzz at 20!

all you *ever* wanted to know about HarfBuzz and then some

behdad

a coffe-table deck for your skimming pleasure—January 2026

Agenda

- **HarfBuzz** History and state of the project
- **Rust** Addressing shaping needs in Rust
- **Standardization** Making changes to the font format
- **Moonshot** Beyond incremental advancements

HarfBuzz

C++ platform with a C library API
and accompanying cmdline tools

HarfBuzz

/hærfˈbɒːz/

From Persian حرف (***Harf***: letter) and باز (***Buzz***: open).
Transliteration of the Persian calque for *OpenType*.

noun

Industry-standard Open Source *text shaping* engine.

adjective

Insincerely talkative. *glib*. A nod to the project's GNOME origins.

What is HarfBuzz?

- To most clients: a ***text shaper***. That is, figure out what glyph shapes need to be displayed and at what positions, to show legible text on screen or in print, in every **language**, **writing system**, and **font**.

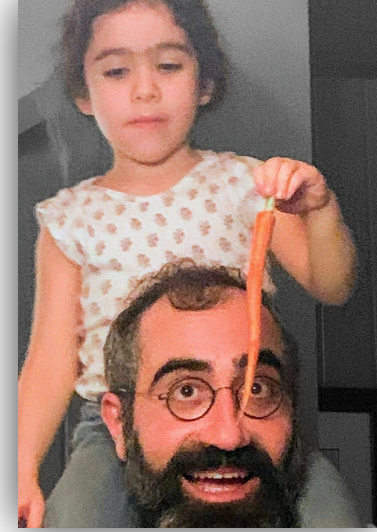
But a lot more, over the years:

- **Integration** glue to various platforms (FreeType, Cairo, GLib, ICU, …)
- A font ***subsetter*** and variable-font ***instancer***. Built on top of the same infrastructure, compiled to `libharfbuzz-subset`.
- More recently: **draw** (monochrome) and **paint** (color-fonts) APIs. No hinting support. Bring your own rasterizer.
- Text-shaping runtime ***tracing*** infrastructure as used in *Crowbar*, a shaping debugger by Simon Cozens, running in the browser on WebAssembly.

History

- **Started** in 2005 by Pango & Qt people to merge the text shaping engines.
- **Moved** to *freedesktop.org* to signal a cross-desktop shaping solution.
- I started a complete **rewrite** in 2006, which is the current HarfBuzz.
- Designed around a few foundational decisions, some of which required **C++**.
- Still, only exporting a clean and ergonomic **C API**.
- No *so-version* jump nor **API/ABI** break since ~2012. Neither on the horizon.
- **Replaced** three Open Source shaping engines: *Pango*'s, *Qt*'s, & *ICU Layout*.
- **Defensive** codebase with novel error-handling patterns that mitigate risk.
- *Extremely low* # of CVEs. Mostly DoS. Barely any crashers & no exploits.
- Code-base is ~100k lines of code. The text shaper is just a fraction of that.
- **Low bus-factor** 🙈. Mostly Khaled Hosny (maintainer) and I (primary developer) plus ~three occasional domain experts.

Vision: the `ffmpeg` of text shaping

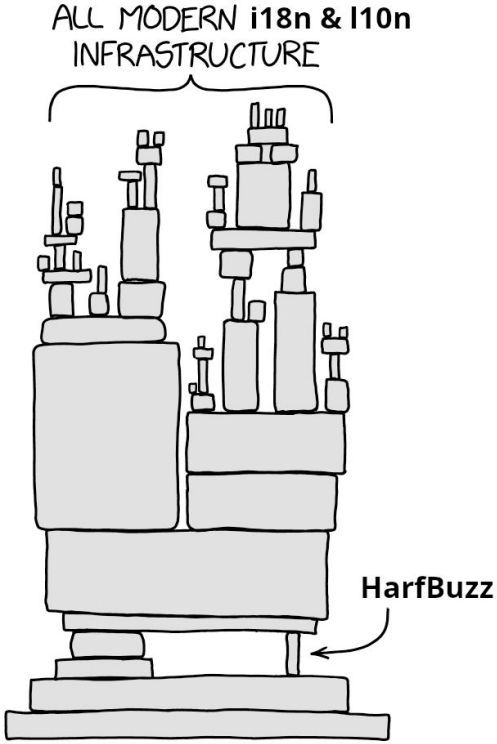


Ubiquitous, pragmatic, understands everything, survives bad input.

Adoption strategy (in hindsight):

- **Market-share** is king.
- Embrace **Open Source**.
- **Permissive license** to maximize reach (old-MIT).
- Linux desktop was *never* gonna be enough.
- Become the **industry standard** for text shaping.
- If we do our job **right**, nobody will notice.
- **Performance** that entices adoption.
- Unmatched **agility**. Zero-day Unicode updates.
- **Robustness** trumps **correctness**, which trumps **performance**. Achieve **all**.
- **Leverage** broad adoption to responsibly experiment at the font-format layer.

HarfBuzz shapes majority of text on modern screens.



Adoption: Open Source

- **Operating systems:** Android, ChromeOS, Linux distros (Fedora, Ubuntu, …)
- **Browsers:** Chromium, Firefox, WebKitGTK, Chromium derivatives…
- **Desktops:** GNOME, KDE, EFL, Flutter, GTK+ / Qt derivatives…
- **Apps:** LibreOffice, XeTeX, Inkscape, GIMP, VLC, VS Code & other Electron apps, Emacs, many others…
- **Programming languages:** Official OpenJDK Java implementation, .NET API (`HarfBuzzSharp` namespace).

Exceptions:

- ***Wine*** remains the only major Open Source platform to use its own text shaper. Porting to HarfBuzz is theoretically possible, but no one has tried.

Adoption: Proprietary

Many major proprietary platforms use HarfBuzz:

- **Google**’s Chrome, text-ads backend in the data-centers.
- **Adobe** Creative Suite: Photoshop, Illustrator, InDesign.
- **Microsoft** Edge, .NET, VS Code.
- **Amazon** Kindle & FireTV.
- Gaming consoles & engines.
- Smart TVs, car displays.
- Smart watches, VR headsets, other consumer gadgets .
- **Figma**, **Canva**, **Instagram**, etc, etc.
- Almost anywhere that cross-platform text layout is needed.

HarfBuzz: Correctness

- In the initial decade, trying to match Microsoft ***Uniscribe*** / ***DirectWrite***.
- Systematically compared to Uniscribe across a **Wikipedia** corpus. Joint work with Jonathan Kew of Mozilla and Roozbeh Pournader.
- Diverging from Microsoft when warranted and only to improve, *not* **laziness**.
- ***Rule of 2 out of 3***: When Microsoft **DirectWrite** and Apple **CoreText** disagree, side with one of them. Or improve on both.
- Over **6,000** shaping tests. Mostly developed in-house over 15 years.
- Heavily **fuzzed** for robustness, by *oss-fuzz* and Chrome fuzzing infrastructure.

Thanks to billions of users, HarfBuzz is now its own reference point.

market share

is

HarfBuzz: Design

- Robust, portable, user-friendly, correct, fast, memory-efficient, thread-safe.
- For fifteen years we resisted adding *barely any* caches whatsoever.
- Pluggable *unicode-funcs* and *font-funcs* make integration with existing Unicode & graphics systems a breeze.
- Alternative *backends* (Microsoft Uniscribe / DirectWrite, Apple CoreText, etc) make correctness & performance testing & comparisons effortless.
- Sanitization of `mmap`ed font then unchecked access of overlaid structs.
- HarfBuzz as the *universal shaping API*: On iOS, disable the HarfBuzz native shaper and bridge it to Apple's system CoreText library, saving 100s of kilobytes in mobile app binary size, while shipping the same HarfBuzz-using shared client code with Android. As seen in a *major* photo social-network app.

HarfBuzz: Codebase

- *Opinionated* coding. Loved by us, hated by others. Steep learning curve.
- Subset of C++11. *SFINAE*, *CRTP*, & other C++ patterns.
- Operator-overloading for transparent endianness conversion.
- Templates for code reuse.
- Custom subset of C++20 Ranges for functional pipelines.
- State-machines generated using `ragel`. Data tables packed using `PackTab`.
- Walking on the edges of *undefined-behavior*. Memory barriers to the rescue.
- Lock-free synchronization: atomic ints / pointers & `cmpexch`.
- Defensive coding: no `nullptr` or dangling-pointers. NULL regions & *Common Region for Access Protection* (CRAP).
- No required dependencies, not even the C++ STL. Many integration bridges.
- Little tech debt.

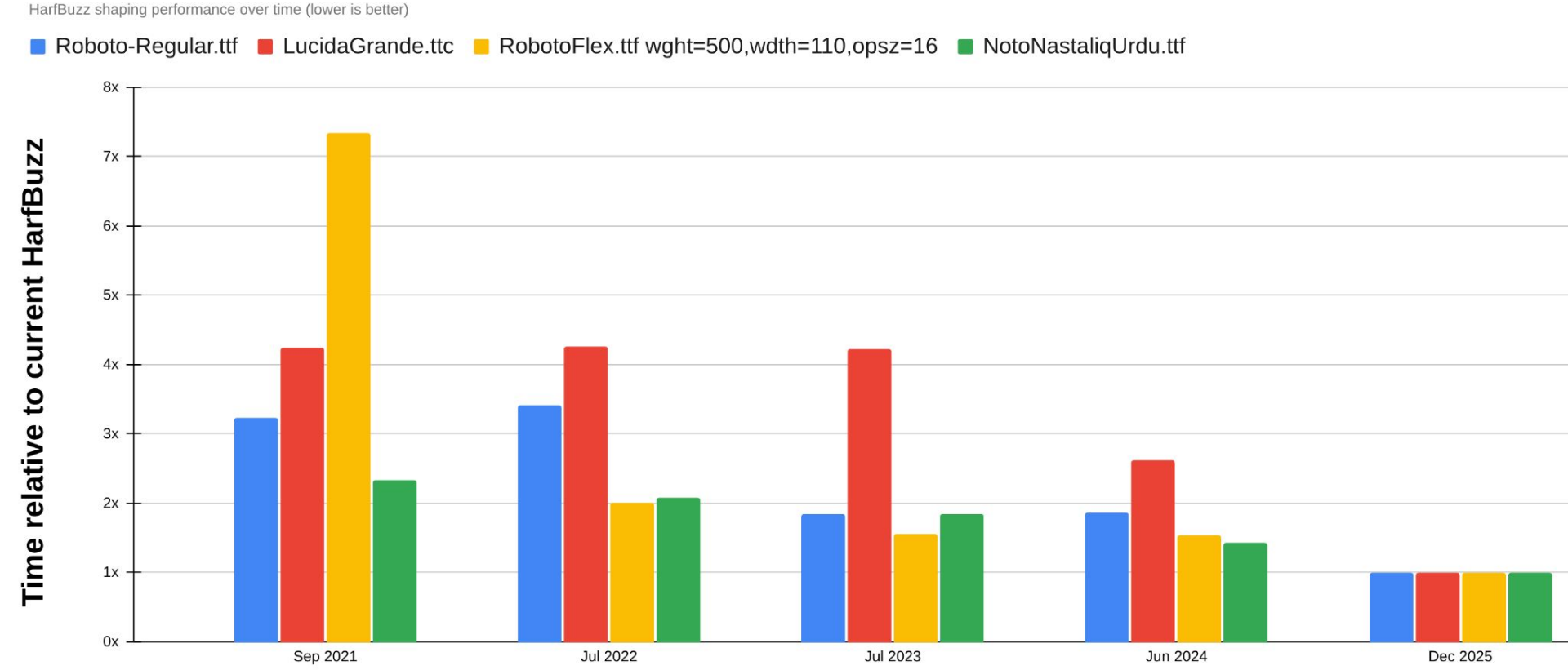
HarfBuzz: Performance

- Consistently most performant shaping engine in its class [discuss `kbts`]. 🏁
- Various novel data-structures, algorithms, and *fast-path*s at work.

Over the past five years:

- Trade kilobytes of cache memory per font file for significant speed gains.
- All caches are shared across threads with lock-free synchronization.

HarfBuzz: Performance improvements over five years



HarfBuzz: Core data-structures

In `shape`:

- **`hb_set_digest_t`** Fast negative filter for set of integers with locality.
- **`hb_cache_t`** Fixed-size bucket-based LRU int-to-int cache.
- **`PackTab`** Compact int-to-int mapping table generator.

In other parts of the library (`subset`, `draw`, `paint`, …):

- **`hb_set_t`** Sparse set-of-ints with locality.
- **`hb_map_t`** Custom hashmap.
- **`hb_decycler_t`** Lightweight DFS cycle detector.

HarfBuzz: Command-line tools

The font-engineer’s Swiss Army knife for text shaping & more.

- **`hb-shape`** The core of HarfBuzz shaping, from the command line, with all kinds of knobs and bells and whistles, producing textual / JSON output.
- **`hb-view`** Use Cairo graphics to render shaped text, including color fonts.
- **`hb-info`** Inspect fonts and extract various information.
- **`hb-subset`** Font *subsetter* and variable-font *instancer*, useful during font production, web-font serving, and printing.

An ecosystem of bindings and ports

Official supported on **`github.com/harfbuzz`**:

- **HarfBuzz:** C++ platform with a C API.
- **HarfRust:** High-fidelity Rust shaper port.
- **uharfbuzz:** Ergonomic Python bindings.
- **harfbuzzjs:** JavaScript & WebAssembly.

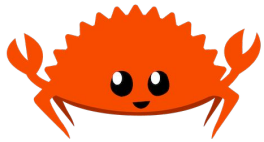
Numerous other bindings and ports too:

- **HarfBuzzSharp:** Part of .NET API.
- **ImageSharp:** Native .NET port.
- **go-text/typesetting:** Go port.
- **luaharfbuzz:** Lua bindings.
- **gobject-introspection:** Clunky auto-generated bindings.

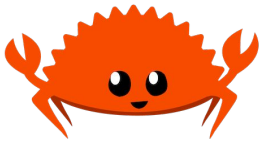
Rust

addressing *safe* shaping
prior art & our offering

`harfbuzz_rs`: Safe high-level bindings to HarfBuzz

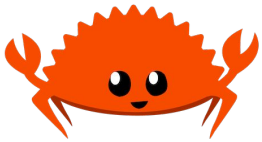


- Started in 2017 by Manuel Reinhardt.
- A high-level safe Rust API to HarfBuzz.
- On top of the `harfbuzz-sys` bindings.
- `harfbuzz-sys` bindings developed by Servo.
- Little adoption otherwise.



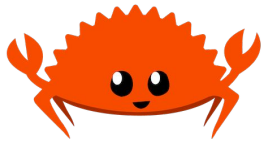
Allsorts: A clean-room text shaper implementation

- Started in 2019 by *YesLogic*, the creators of *Prince* HTML-to-PDF commercial solution.
- Contracted *Nate ‘n8willis’ Willis* to document what HarfBuzz does in its *complex* shapers.
- Clean-room implementation solely based on the OpenType spec and Nate’s produced docs.
- Using custom domain-specific language for font data parsing.
- Still active but incomplete: eg. missing the *Universal Shaping Engine* component, used by dozens of South-East Asian and other writing systems.



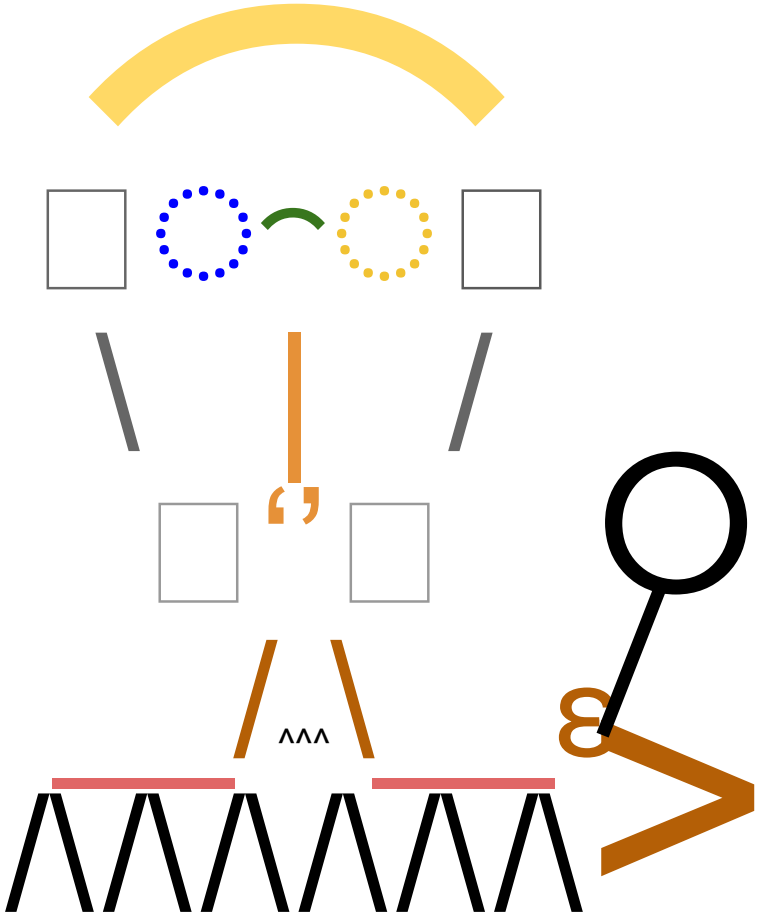
RustyBuzz: A fully safe port of HarfBuzz to Rust

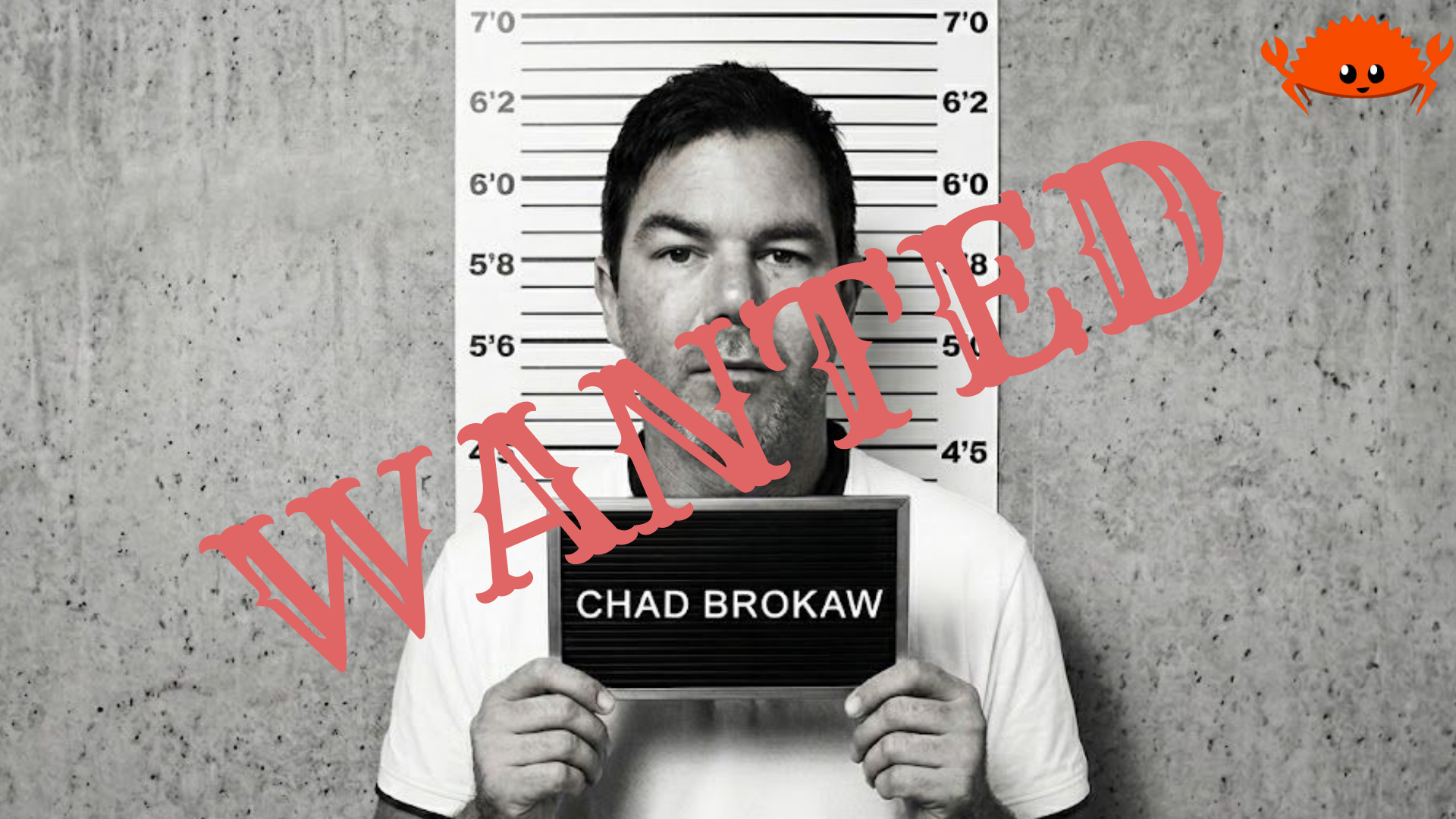
- Started in 2020 by the prolific *RazrFalcon*.
- Loyal port of HarfBuzz's shaping logic to fully safe Rust.
- Originally needed by RazrFalcon for `resvg`.
- Based on the `ttf-parser` crate (*RazrFalcon*, 2019) for reading font data.
- Was quickly and widely adopted, by `cosmic-text`, Typst, others.
- Deprecated by RazrFalcon in 2023.
- *Laurenz Stampfl* took over updating it to latest HarfBuzz code.
- Currently stale, matching HarfBuzz 10.1.0 from November 2024.
- For all realistic purposes, deprecated & unmaintained.



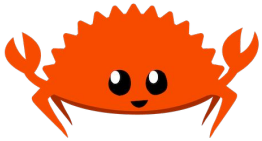
Swash: Because *why not*

- Started in 2021 by *Chad Brokaw* with great ambitions.
- Chad Brokaw's implementation from scratch because he's Chad Brokaw.
- Initially used in Chad Brokaw's Parley text layout engine.
- Chad Brokaw got offered a job by Google Fonts.
- Fortunately for the ecosystem, Chad Brokaw abandoned Swash.
- Chad Brokaw still addressing the Rust safe shaping need, at Google.




7'0
6'2
6'0
5'8
5'6
4'5
CHAD BROKAW
WANTED

Google Oxidize

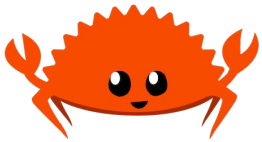


“Wherein we contemplate moving shaping, rasterization, font compilation, and general font inspection and manipulation from Python & C++ to Rust.”

—*Rod Sheeters*

Tech-lead & manager, Google Fonts

January 9, 2022

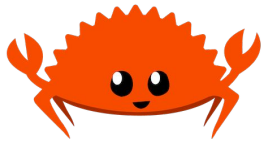


Google Oxidize

- Multiple full-time Google Fonts engineers allocated to the effort.
- *Fontations* set of crates: `font-types`, `read-fonts`, `write-fonts`, `skrifa`, `klippa`, …
- Use procedural macros to codegen font table compilation / decompilation code from semantic declarations.
- `skrifa` is the shape extraction library, replacing FreeType. Supports hinting: both font-side hinting & FreeType-style autohinting. Bring your own rasterizer.
- `fontc` replaces Python `fontmake` as the font compiler from various sources.
- Active and high-speed development.
- Text shaping *was* left at TBD.

HarfRust

one crate to rule them all

HarfRust

/hærfˈrɒːst/

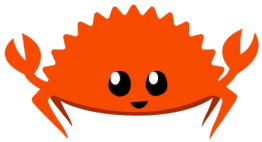
From Persian حرف (***Harf***: letter) and راست (***Rust***: true / straight).
Transliteration of the Persian calque for *TrueType*, the precursor to OpenType.

noun

The same trusty Open Source *text shaping* engine, in fully safe Rust.

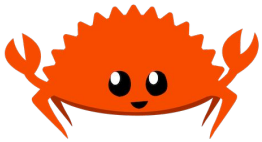
not to be confused with

RustType, a font rasterizer in Rust. Small fraction of *FreeType* functionality.



HarfRust: Safe, correct, and performant Rust port

- Born in 2024 as a fork of RustyBuzz by Chad Brokaw at Google Fonts.
- First (0.1.0) release in June 2025.
- Use Oxidize's `read-fonts` crate instead of unmaintained `ttf-parser`.
- Up to date with latest HarfBuzz shaper code.
- Always in sync with latest HarfBuzz release moving forward.
- Chad Brokaw as the lead & maintainer. I am a contributor.



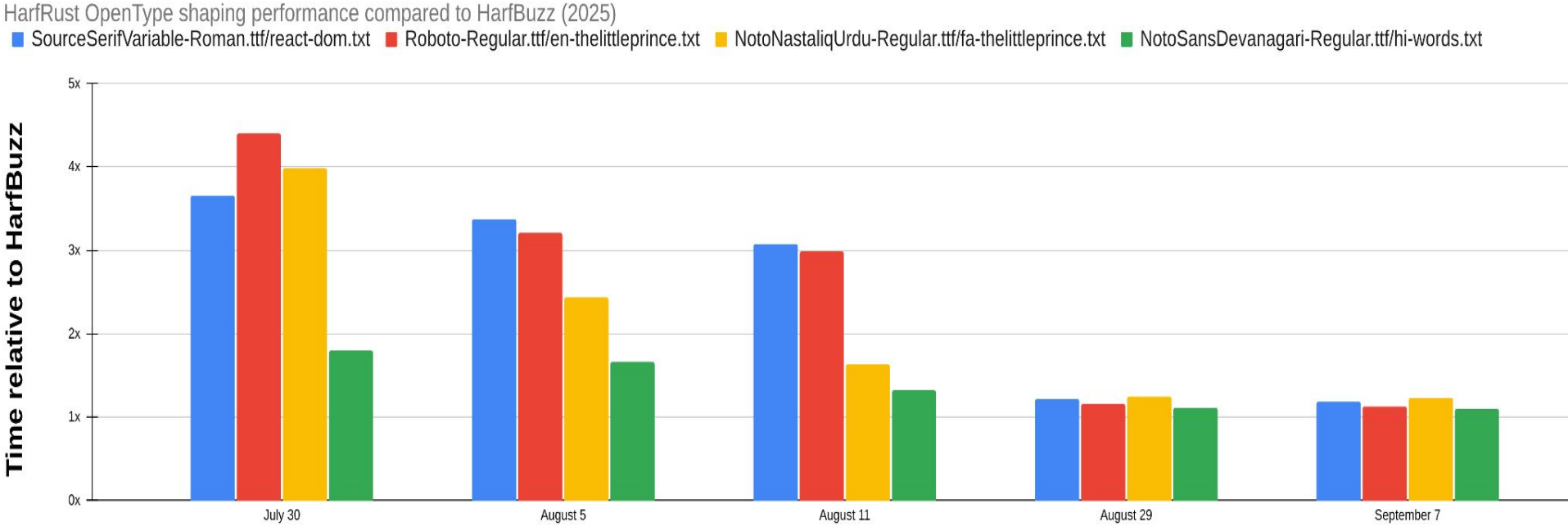
HarfRust: Notable users

- The venerable `cosmic-text` crate
- Parley text layout engine by Chad Brokaw, et al.
- Blitz browser
- Canva. They recently hired Raph Levien and are heavily invested in Rust graphics & text rendering stack.
- Several systems slow to jump at HarfRust, as C++ HarfBuzz mature & stable.
- **Soliciting commitment from other suitors.**
- **We will help you with *your* integration.**

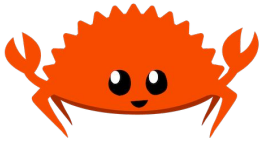
HarfRust: Correctness

- Fails **18** out of **6232** HarfBuzz shaping tests.
- Remaining fails are of legacy fonts or obscure historical cases that might never be needed / addressed.

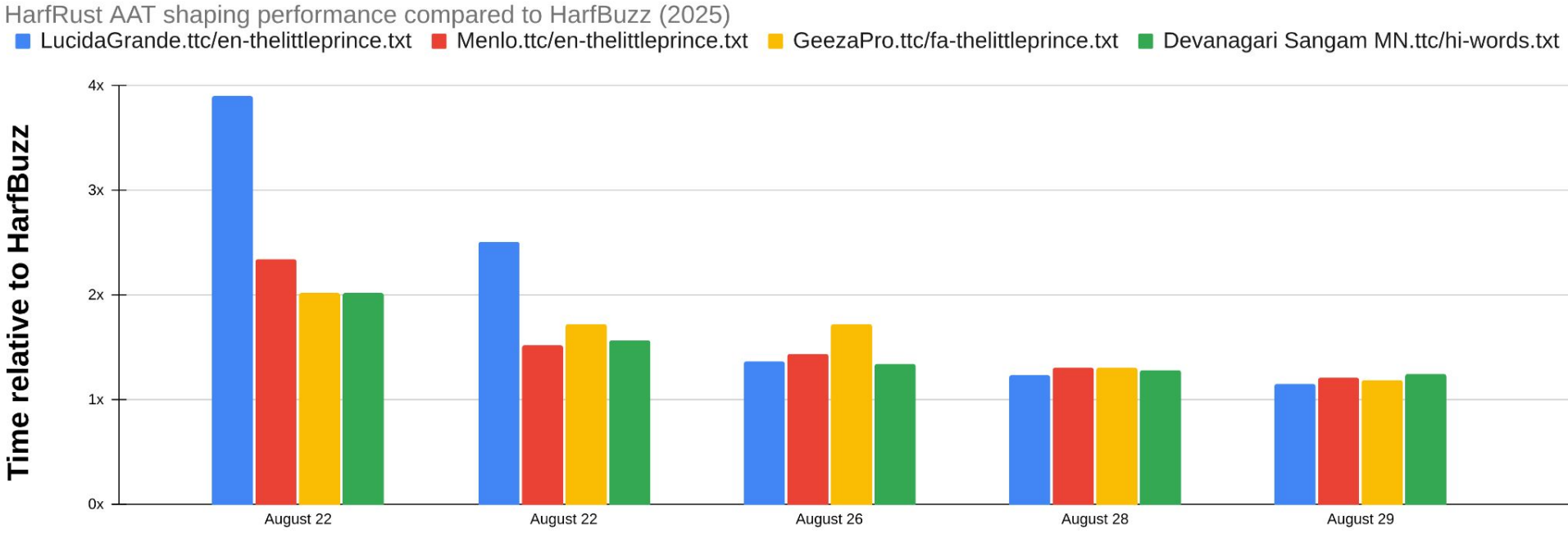
HarfRust: Performance: OpenType

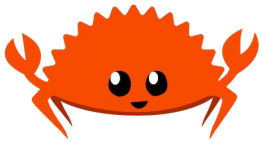


HarfRust: Performance: AAT



AAT is Apple's alternative to OpenType shaping, which while semi-retired, is still used in many Apple platform system fonts and hence important to browsers.



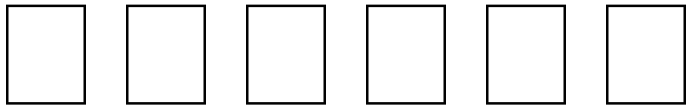


HarfRust: Performance Analysis

- All algorithmic and caching schemes from HarfBuzz ported.
- From 4x slower than HarfBuzz, down to <25% slower on key benchmarks.
- The gap has regressed since 🙈: different architecture for reading font data.
  - HarfBuzz does a *pre-sanitization* pass over the font table data, then uses unchecked access to clean `mmap()`ed pages during shaping.
  - HarfRust uses Fontations' codegen-ed structs & Rust checked array access.
- New optimizations frequently speed up HarfBuzz more than HarfRust.
- We are exploring making Fontations reading faster and embrace the pre-sanitization model. Early results are very promising.
- "C++ is still perf king for those who know how to hold it" —*Chad Brokaw*

Standardization

extending the font format



Font format as bottleneck

In 2014 while at Google, I noticed that:

- Google was commissioning the production of internationalized Noto fonts to Monotype and releasing those as Open Source fonts.
- Google ships such fonts in Android, and to Chrome via Google Fonts service.
- Android and Chrome use Open Source libraries HarfBuzz and FreeType, for text shaping and drawing respectively.
- The only part that we *didn't* have control on was the font format.

If we can extend the font format, we can unlock new kinds of innovation.

Font format innovation path: OpenType

- OpenType is the dominant font format specification.
- OpenType is property of Microsoft.
- Industry-wide collaborations with Microsoft have produced impressive results in the past: *variable fonts* and *COLRv1* for example.
- However, Microsoft's willingness to incorporate new changes in OpenType is subject to their engineering and editorial resource availability, which has *allegedly* been going down over the years.
- High chance of impasse. A different strategy is needed.
- After the *color fonts* snafu, Google reluctant to ship its own format extensions.
- Unless **standardized** or **shipped by another major player** first.
- In 2023, Apple silently shipped our proposed *avar2* technology. Chrome still *doesn't*.

Font format innovation path: ISO Open Font Format

- In mid 2000s, to curb competition, Microsoft donated a copy of OpenType to ISO for standardization.
- By 2007, ISO *Open Font Format* (*OFF*), formally known as ISO/IEC 14496-22 was born.
- Microsoft maintains OpenType as a separate specification to this day.
- ISO contribution process is slow, painful, and based on an “ad-hoc” working group’s consensus.
- The ad-hoc group’s recommendations are then voted on by the national delegations.
- The whole process takes between three to five years, from proposal to released standard.

Font format innovation path: ISO Open Font Format

- Chrome is reluctant to ship proposals before late stages of standardization.
- Android is reluctant to ship font format extensions not supported by Chrome.
- Chrome can be open to supporting what other platforms already support.
- This puts us in a double-bind: if we innovate in the file format, we either have to wait 4 years before Chrome will ship it, or hope that Apple would go ahead and ship it (and with no announcement for that matter).
- While in the meantime, Microsoft and Apple feel free to ship font format extensions in their products, and then *maybe* submit to ISO.
- This is a structural problem, not a failure of any single party.

Font format innovations: a timeline

- **1997** OpenType 1.0 was released by Microsoft and Adobe, bringing to an end the *Font Wars* of the 1990s. Documentation leaves a lot to be desired.
- **2000s** Microsoft support Indic scripts in *Uniscribe*, their shaping engine.
- **2007** *ISO Open Font Format* is released, as a derivative of OpenType.
- **2009** Introduction of Microsoft's proprietary *DirectWrite* as successor to Uniscribe, as *de facto* shaping standard.
- **~2012** Apple, Google, Microsoft, and Adobe+Mozilla each invented a different way to support color fonts. All four find themselves standardized at the ISO Open Font Format as competing techs.
- **2014** I organized an informal OpenType face2face meeting with Microsoft, Apple, Adobe, other experts to advance Indic-like text shaping.

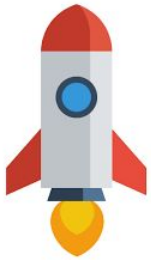
Font format innovations: a timeline

- **2015** Microsoft announced *Universal Shaping Engine* (USE), a more systematic and advanced version of HarfBuzz's homegrown *South-East Asian* shaper. HarfBuzz implemented USE quickly and HarfBuzz 1.0.0 was released on July 26, 2015.
- **2016** Based on industry-wide interest, Microsoft invited Adobe, Apple, and Google to collaborate together on what became *OpenType 1.8 Variable Fonts* on September 14th, 2016. The Open Source stack & Chrome are first to ship.
- **2017** Microsoft reduced resources to OpenType and font software drastically, assigning *Peter Constable*, the OpenType editor, to other projects.
- **2019** With *Dominik Röttsches* and others at Google, we proposed *COLRv1*. With Microsoft buy-in, the work was added to OpenType.

Font format innovations: our roadmap

- **2022** I proposed a set of changes collectively called *Boring-Expansion* to the ISO Open Font Format, with Google's financial support and backing. This is an incremental extension of the font format to address decades-long limitations. Other parties participated in technical discussions. The final result is due to be released as ISO spec in the first half of 2026. 🏁 [was it?]
- **2026** Started next phase of *Boring-Expansion*: new proposals to significantly speed up the slowest text shaping needs, code-named *layout-ng*.

Moonshot



WebAssembly *in* the font

Beyond-Emulation: WebAssembly for President!

- Time to *really* put the *smarts* into the *fonts*.
- Current code-*vs*-data boundary is quite arbitrary:
  - **Font file** provides *data* for how to convert from Unicode text into positioned glyphs.
  - **Layout engine** controls the layout based on font shaping result & Unicode properties. Sometimes with full programmability, as in *LuaTeX* or *SILE* typesetting systems.
- *Securely* inject programmability into the font file.
- *WebAssembly* already portable assembly format with tooling.
- *Generic* solution to many hard shaping problems.
- No *specific* shaping engine changes needed!
- Another use-case: *Graphite*, *DecoType ACE*, alt. shapers.

WebAssembly in smart-fonts! Parallel precedents

- OpenGL 1 fixed-pipeline	→	OpenGL 2+ / WebGPU *programmable* shaders
- Static web-servers	→	CGI, Perl, PHP, Python, JS… *scripting*
- CGI, etc, web-server scripting	→	WebAssembly in edge-computing
- Static web content	→	JavaScript / WebAssembly programming
- Linux physical servers	→	Linux as container of *many* virtual servers
- Pre-digital typesetting	→	METAFONT, TeX, LuaTeX

Beyond-Emulation: BIG question

Beyond-Emulation: the elusive Nastaliq style of Arabic

Requires collision-avoidance or approximation of:

- Easy to code in eg. C / Rust.
- Hard to express as data rules.

OpenType
(No mitigations)

OpenType
(Best effort)

WASM

Beyond-Emulation: Arabic typesetting

Many alternative models developed over the decades:

- *Graphite*
- *DecoType*
- *DigitalKhaatt*

Other custom-built solutions, some only available in Iran or Pakistan.

- We don’t *want* to standardize one solution. It would be way too *specific* for a *generic* font format.

Beyond-Emulation: SIL Graphite

SIL Graphite is an alternative shaping model to OpenType Layout.

Only Firefox, LibreOffice, XeTeX, and possibly minor clients support it, by configuring HarfBuzz with the extra `libgraphite2` backend.

Instead: compile `libgraphite2` to WebAssembly to embed in the font.

- 90kb `Wasm` table payload (40kb as WOFF2 compressed).
- 3x slower (with JIT) than natively compiled library.

Beyond-Emulation: how it looks like

`Wasm` table in the font includes the shaping bytecode. Currently HarfBuzz-centric.

```
#define HB_WASM_INTERFACE(ret_t, name) __attribute__((export_name(#name))) ret_t name

#include <hb-wasm-api.h>

bool_t
shape (void *shape_plan,
       font_t *font, buffer_t *buffer,
       const feature_t *features, uint32_t num_features)
{
  if (!shape_with (font, buffer, features, num_features, "ot"))
    return false;

  // Modify buffer;

  return true;
}
```

Beyond-Emulation: how it works

- Proof-of-concept experimental implementation in HarfBuzz.
- Using the nimble `bytecodealliance/wasm-micro-runtime` C API.
- The runtime lacks basic security features like *timeboxing*.
- WebAssembly specification still *very* heavy-handed on instances:
  - Memory allocations happen in multiples of 64kb.
  - No virtual address-space or memory mappings between the host and the guest.
  - Multiple address-spaces envisioned but not fleshed out in the spec.
  - Acceptable performance *unachievable* without runtime instance sharing.
- Rust might have more battle-tested WebAssembly host runtimes.
- Fonts have been built both from C / C++ and Rust shaping logic code.
- `https://github.com/harfbuzz/harfbuzz-wasm-examples`
- `llama.ttf`, `translate.ttf`, etc.

Beyond-Emulation: where to go from here

- Research-level project for the foreseeable future.
- Might never happen in browsers & operating-systems.
- Maybe just enabled by trusted **desktop-publishing** systems.
- Rewrite the WebAssembly shaping API to be more independent of HarfBuzz internal data representations.
- Assess implementing in HarfRust using a Rust-based WebAssembly host.
- Viewed another way: a HarfBuzz-originated text shaping ***plugin*** **system**.
- Ultimately: a solution to the ***font format extension*** **bottleneck**.