

What Kaomoji Are Made Of, and Where They Break in Basic Fonts and Markdown

A kaomoji looks like one picture, but it is a short string of characters, often borrowed from several writing systems. This study takes apart the 1,616 faces our site publishes, tests which ones a basic set of fonts can draw, and checks what copying, pasting and Markdown do to each.

MAIN FINDING

Copying a face never altered one in our browser tests, but whether you see it depends on fonts: with basic fonts only, 61% of faces had a character the fonts could not draw. Standard Markdown changed 4% to 6% of faces, depending on the engine and where the face sits.

- Unicode 18.0 breakdown, Linux font tests and Markdown tests complete
- Not tested on Windows, Mac, iPhone, Android or inside any chat app
- How people read the faces was not studied

PUBLISHED	VERSION	EVIDENCE
October 16, 2026	1.0	Unicode data files, our own collection pages, headless Chromium 141 on Linux, three Markdown engines. No people took part.

Canonical URL: <https://ultratextgen.com/research/kaomoji-unicode-rendering-study/>
UltraTextGen Research · Version 1.0 · Published October 16, 2026

SUMMARY

What the study found

1,616

different kaomoji published on our site

They come from 7,085 placements: 4,955 on 187 collection pages, the rest in kaomoji sections of other pages, face-shaped tiles elsewhere, the generator and the decoder. Translated pages reuse faces, so each distinct string is counted once.

• MEASURED FROM OUR PAGES

23.6%

of faces mix two or more writing systems

382 faces. Greek letters next to Japanese katakana or hiragana is the most common pairing (36 and 35 faces). Faces borrow characters for their shape, not their script.

• MEASURED FROM UNICODE DATA

1,616 of 1,616

faces came through copy and paste unchanged

We inserted each face (no key presses) into a text box, a text area and an editable region, and passed it through the browser clipboard inside one page, JSON, a web address and local storage in Chromium. Only two faces changed, and only in how an editable region reports its spaces.

• MEASURED IN A BROWSER

61%, or 21%

of faces had a character the test fonts could not draw

With basic fonts only, 986 of 1,616 faces lost a character. With two Japanese fonts (IPAGothic and its proportional sibling IPAPGothic) and one Chinese font (WenQuanYi Zen Hei) added, 334 did. Faces using another writing system fared worst: 92% against 39%.

• MEASURED, LINUX FONTS ONLY

5.3%

of faces were altered by a standard Markdown engine

86 faces in the CommonMark reference engine when the face stood alone: backslashes eaten, asterisks and underscores turned into italics, a leading > turned into a quote. Across three engines and three placements (the third gave the same result as a face alone) the range is 4.1% to 5.8%.

• MEASURED, THREE MARKDOWN ENGINES

49.9%

of faces would change under Unicode compatibility normalization

That is the NFKC form that some tools apply to text. It would merge 88 distinct faces into 43. The gentler NFC form changes none.

• MEASURED FROM UNICODE DATA

How to read these numbers

The faces are the ones this site publishes, so the figures describe our collection, not every kaomoji in the world. The font results come from Linux fonts we chose, which is a test of what fonts cover, not a share of devices. We did not test Windows, Apple or Android systems or any chat app, and we did not ask anyone how they read a face.

WHAT TO DO

What this means for you

- **Need a face that draws with the fewest fonts?** In our tests plain keyboard characters were the safe choice: all 166 faces built only from ASCII characters, such as (^_^) and (T_T), drew with the basic fonts. They are not safe from Markdown: 16 of them changed, mostly because a leading > starts a quote. The faces that failed most often in basic fonts used Japanese punctuation and halfwidth katakana (・ノ。ー、), which need a font with Japanese coverage, or characters from Thai, Gujarati, Tibetan or Yi.
- **Pasting a face into a chat or post that reads Markdown?** Four characters cause nearly all the damage: the backslash (33 faces contain one, 28 changed), the asterisk (84 contain one, 28 changed), the underscore (220, 20 changed) and a leading > (all 21 changed). The shrug ^_(`\\_)/^ loses its backslash. Typing three backslashes before the underscore, ^\\\\_(`_)/^, brought it back in all three engines we tested, and so did wrapping the face in backticks, which shows it unchanged in code formatting. Apps with their own Markdown rules may differ, and we did not test them.
- **Building a tool that handles faces?** Keep the exact string the person gave you. Normalizing with NFKC would change half of them and merge some into others. Which Unicode version your language ships

also matters: the same character can be classed differently between versions (see normalization).

- **Seeing empty boxes where a face should be?** The box is drawn by your device because none of its fonts has that character. The guide to why text shows as boxes explains the three kinds of failure and how to tell them apart. The Unicode rendering study shows how fast new characters reach fonts.

SCOPE

What we counted

A kaomoji here is a face that appears as a copy tile on our site: on a collection page, in a kaomoji or emoticon section of another page, as one of the generator’s sixteen starter faces, or as one of the forty named faces in the decoder. A tile whose text is built like a face but sits on some other page also counts, when it has a pair of brackets, a body between them, no emoji picture and no ordinary words. Pictures alone, such as a bunny emoji, are not faces. Faces the generator could combine but that nobody published are not counted.

WHERE A FACE WAS FOUND	PLACEMENTS
Collection pages (187 pages in 17 languages; the English ones hold 1,300 distinct faces)	4,955
Kaomoji or emoticon sections on other pages	1,824
Face-shaped tiles elsewhere on the site	250
Generator starter faces	16
Decoder named faces	40
All placements	7,085
Distinct faces (byte for byte)	1,616

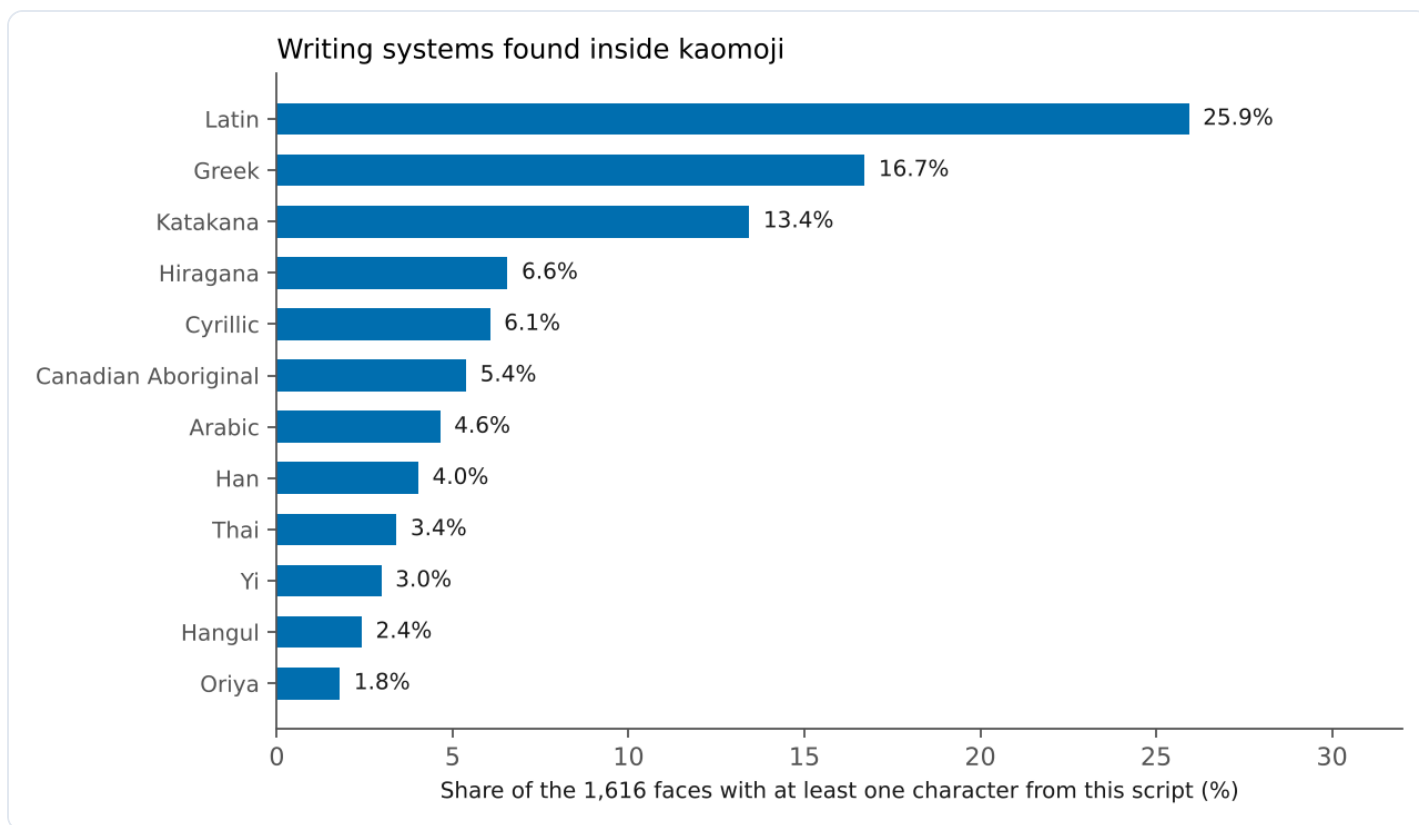
Faces are shared between languages, so the 7,085 placements are not independent observations; every figure on this page counts each distinct face once unless it says otherwise. We took the list from the site’s source files at one commit and rebuilt it on a later commit with identical results.

RESULTS

Writing systems inside faces

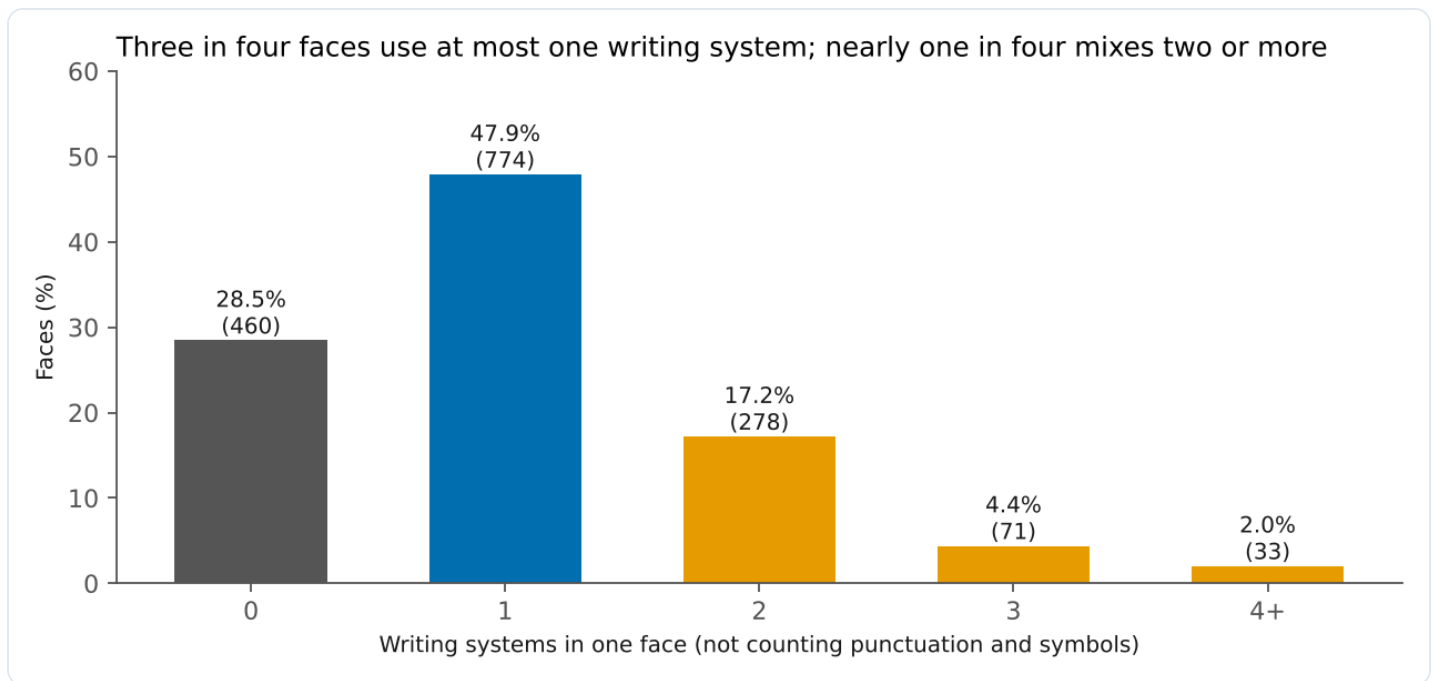
Unicode gives every character a script, such as Latin, Greek, Katakana or Thai, plus a catch-all called Common for punctuation and symbols and one called Inherited for marks that attach to another character.

The 1,616 faces use 39 of these and 83 of Unicode's character blocks, with 692 different characters in total. 97.1% of faces contain at least one Common character, because that is where brackets, dots and most symbols live.



Faces containing at least one character from each script, Common and Inherited left out. Latin (25.9%), Greek (16.7%) and Katakana (13.4%) lead.

Most faces stay inside one writing system, but 23.6% do not. Greek and Cyrillic letters often stand in for eyes (ω , μ), katakana and hiragana supply arms and cheeks, and Thai, Tibetan, Arabic and Canadian Aboriginal characters appear for their shapes. One face uses six scripts.



Number of writing systems per face, not counting Common and Inherited. 460 faces use none, 774 use one.

A Script label is not an origin

A Katakana or Thai label says which block of Unicode a character belongs to. It does not say where the kaomoji was invented or that the face is Japanese or Thai. Unicode's data also shows why plain labels mislead: 21 of the Common characters here, such as the middle dot `·` and the halfwidth full stop `,`, are listed by Unicode as also used with one or more East Asian scripts. They appear in 360 faces. We report scripts as a count of what is in the text and make no claim about where a face came from.

RESULTS

The shape of a face

- **Short.** The median face is 7 characters long, nine in ten are 12 or shorter, and the longest has 34.
- **A short list does most of the work.** 21 characters fill half of all character positions across the faces, 121 fill 80% and 344 fill 95%. A long tail of 233 characters appears in only one face, and 155 occur only once in the whole list. The brackets (`(` and `)` are in 77.5% and 78.3% of faces.
- **Width is not fixed.** 66.1% of faces (1,068) contain a character whose East Asian Width is Ambiguous: Unicode leaves it to the font and language whether the character takes one column or two. 1,038 of them change their column count between the two treatments. The same face can be wider on a Japanese system than on a Western one.
- **Marks and astral characters.** 127 faces (7.9%) contain a combining mark that attaches to the character before it, such as the eyelids in the lenny face. 77 faces (4.8%) contain a character above U+FFFF,

which some older software counts as two characters.

RESULTS

Normalization can merge faces

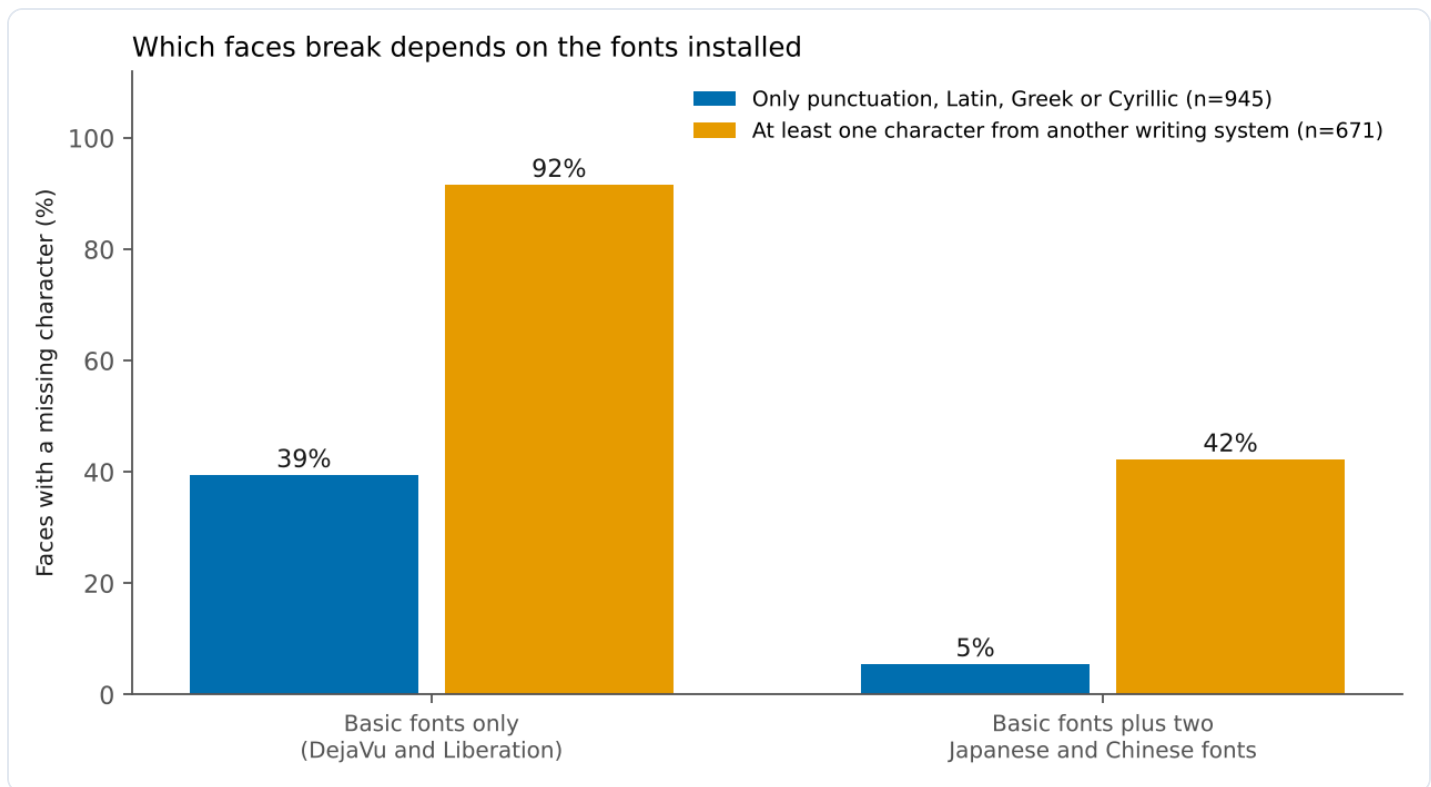
Some tools rewrite text into a standard form before they store or compare it. The NFC form joins letters with their accents and changed none of the 1,616 faces. The NFKC form also folds compatibility characters into plain ones, and it changed 807 faces (49.9%): a little over half through fullwidth and halfwidth forms, which it turns into ordinary ones, and most of the rest through spacing accents such as ´ and ˘, which it turns into a space plus a combining mark. That matters if you deduplicate faces: 88 distinct faces would collapse into 43. For example `\ ( ^ ▽ ^ ) /`, `\ ( ^ ▽ ^ ) /` and `\ ( ^ ▽ ^ ) /` become one face under NFKC, although they look and measure differently.

The version of Unicode behind your tools can change answers too. The character 𐄁, one side of the bear face 𐄂•••?, is a lowercase letter in the Unicode 15.1 tables that Python 3.13 ships and an “other letter” in Unicode 18.0, which we used. Every count on this page names the version it used.

RESULTS

Which faces break in which fonts

A device draws a character only if one of its fonts has it. We checked every character of every face against three font sets in a Linux container. **Basic fonts** are DejaVu (Sans, Sans Mono, Serif) and Liberation (Sans, Mono, Serif), which stand in for a bare-bones system. **Basic plus Japanese and Chinese fonts** adds two Japanese fonts (IPAGothic and its proportional sibling IPAPGothic) and one Chinese font (WenQuanYi Zen Hei). The third set, every font in the container, includes a near-universal fallback font and could not be validated, as we explain below. We counted a face as broken when at least one of its characters had no glyph in any font of the set.



Faces with a missing character. Basic fonts: 39% of faces without another writing system and 92% of faces with one. With Japanese and Chinese fonts added: 5% and 42%.

FONT SET	FACES WITH A MISSING CHARACTER	FACES USING ANOTHER WRITING SYSTEM	OTHER FACES
Basic fonts	986 of 1,616 (61.0%)	614 of 671 (91.5%)	372 of 945 (39.4%)
Basic plus Japanese and Chinese	334 of 1,616 (20.7%)	283 of 671 (42.2%)	51 of 945 (5.4%)

Faces that use another writing system broke 2.3 times as often with basic fonts (91.5% against 39.4%). The odds were 16 times higher (95% interval 12 to 22). Part of that is plain: katakana needs a Japanese font. So we repeated the comparison after setting aside faces whose only extra scripts are katakana, hiragana or Chinese characters. The odds were still 7.8 times higher (290 of 347 against 372 of 945).

Faces without another writing system broke too: 339 of those 372 broken faces lost a character from the Japanese, Chinese or fullwidth blocks, almost always one that Unicode labels Common, such as the middle dot and the halfwidth full stop. The characters that left the most faces broken with basic fonts were:

CHARACTER	NAME	FACES BROKEN, BASIC FONTS	FACES BROKEN, PLUS JAPANESE AND CHINESE
・	Katakana Middle Dot	91	0
/	Halfwidth Katakana Letter No	83	0
。	Halfwidth Ideographic Full Stop	69	0
—	Fullwidth Macron	69	0
ゝ	Katakana Iteration Mark	61	0
。	Halfwidth Katakana Semi-Voiced Sound Mark	61	0
/	Katakana Letter No	53	0
`	Fullwidth Grave Accent	50	0
・	Halfwidth Katakana Middle Dot	43	0
๑	Thai Digit One	40	40

With the Japanese and Chinese fonts added, the faces that still broke mostly used characters from other scripts, led by ๑ (40), ε (20), 𐄂 (19), 𐄃 (17), 𐄄 (16), 𐄅 (15), 𐄆 (15); they are a Thai digit, Yi radicals used as brackets, a Gujarati digit, a Canadian syllable and Tibetan brackets. The full list is in the characters file. As a rough guide to fallback work, 77% of faces needed two or more fonts to draw in the container with all its fonts, 49% with the Japanese and Chinese set, and 4% with basic fonts.

The container with every font hid the problem

Our container with all its fonts includes one that draws something for almost any code point. The usual check for a missing character, drawing it and comparing it with a code point nobody has assigned, failed its own test there: the unassigned character was not flagged. Reading the fonts' character tables failed the same way. Under the rule we wrote down in advance, the missing-character check is void in that set, so we publish no missing-character rate for it. We can say that seven characters, in 10 faces, are in none of those fonts: the Egyptian hieroglyph fish (U+1319B to U+1319F), U+131FC and U+16910. A real phone or laptop has a different set of fonts, so the two sets above are tests, not forecasts for your device.

RESULTS

Copy and paste

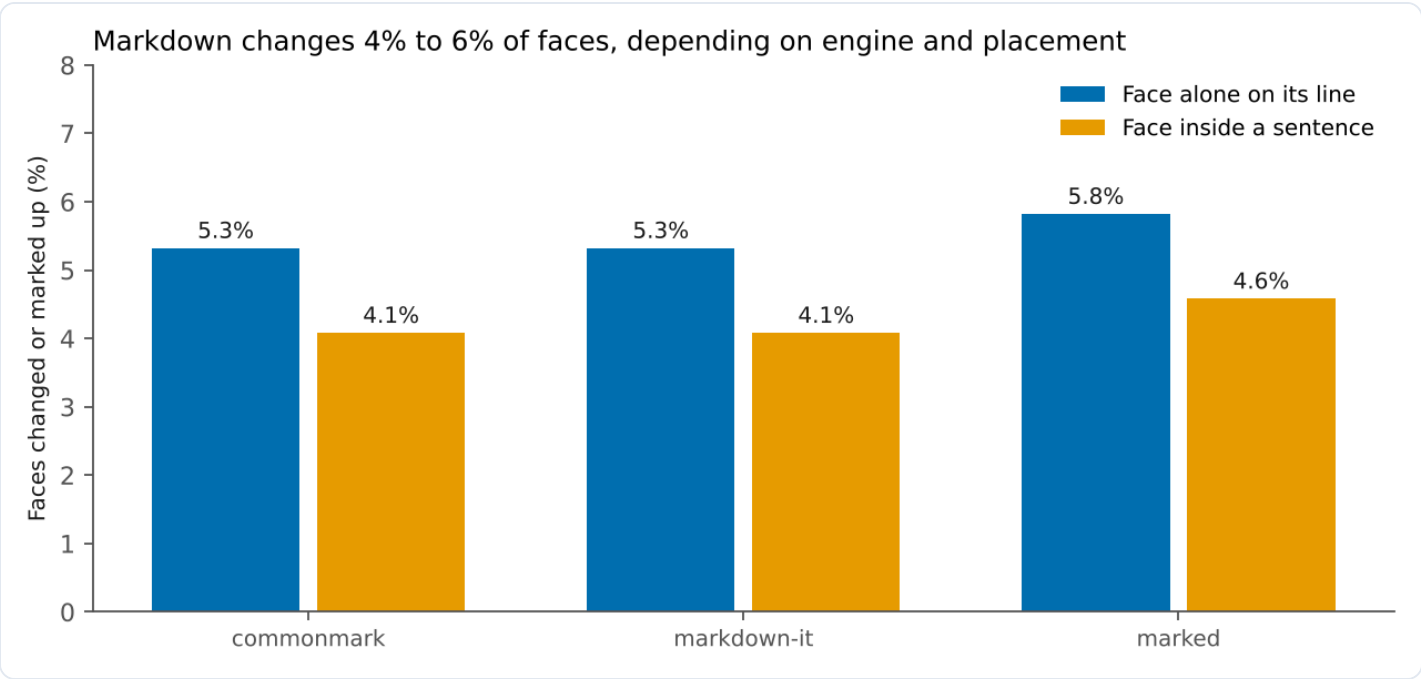
For each face we checked that the string a person types or pastes comes back byte for byte. We used Chromium's text insertion into a text box, a text area and an editable region, a write and read through the browser clipboard inside one page, JSON, web-address encoding and local storage. All 1,616 faces came back identical in every one of them. The one difference is cosmetic: an editable region collapsed the runs of spaces in two faces (`><>` `><>` and `( . _ . ) ?`) when asked for its visible text, while its stored text kept them.

This tests the browser, not the apps people paste into. Text insertion here does not fire key presses, the clipboard test stays inside one page, and a messenger or social app can still rewrite a face after you paste it. Separately, on each of the 404 site pages that carry kaomoji tiles, a browser check found every tile showing the same text it carries for copying (that check script is not in the data folder).

RESULTS

Markdown

Chat apps, forums and note tools often read a small formatting language called Markdown. We passed every face through three engines that follow the standard (CommonMark 0.31.2, markdown-it 14.1.0 and marked 15.0.12) with the face alone on a line, inside a sentence and on the line after other text, and counted a face as changed when the visible text differed or the engine produced formatting. We had made no prediction about this test.



Share of faces changed or marked up by each Markdown engine, alone on a line and inside a sentence, between 4.1% and 5.8%.

ENGINE	FACE ALONE	FACE INSIDE A SENTENCE
commonmark	86 of 1,616 (5.3%)	66 of 1,616 (4.1%)
markdown-it	86 of 1,616 (5.3%)	66 of 1,616 (4.1%)
marked	94 of 1,616 (5.8%)	74 of 1,616 (4.6%)

The two engines that follow CommonMark agree exactly. The damage is easy to predict. A backslash before a punctuation mark disappears (28 of the 33 faces with a backslash changed). A pair of asterisks or underscores turns the text between them into italics (28 of 84 faces with an asterisk and 20 of 220 with an underscore changed). A > at the start of a line starts a quote (21 of 21). A tilde mattered little in CommonMark (1 of 35) and more in marked (9), which reads a pair of tildes as strikethrough. The face (*´ω`\*) comes back as (´ω`) in italics. The most common casualty is the shrug: typed as ^_(`^)\\_/^, it comes back as ^\\_(`^)_/^. Three backslashes before the underscore restored it exactly in all three engines, and so did putting the face in backticks, which shows it unchanged in code formatting. We did not test Discord, Reddit, WhatsApp, Slack or any other app, and their rules differ from the standard.

CHECK

What we predicted, and what happened

Before running the font and copy-and-paste tests we wrote down five predictions, P1 to P5. The text is in predictions-as-written.md, with a list of every place we departed from it. This page cannot prove when we wrote them. The Markdown test, the character counts and the normalization counts had no predictions.

PREDICTION	RESULT
P1. Faces survive copy and paste unchanged	Held. 1,616 of 1,616 in every context we tried.
P2. With basic fonts, faces using another writing system break more often	Held. 91.5% against 39.4%; adjusted p-value below 0.001.
P3. With every container font, fewer than 1% of faces break	Could not be tested. The missing-character check failed its own control in that set, so its results are void.
P4. The drawing check and the second check agree on 98% of faces or more	Held, for a substitute check. They agree on 1615 of 1,616 (basic fonts) and 1616 of 1,616 (with Japanese and Chinese fonts). We had named a browser font query as the second check. It cannot flag a missing glyph, so we read the fonts' character tables instead.
P5. Faces with width-ambiguous characters change width by 10% or more between two monospace fonts more often	Failed, in the opposite direction. With the Japanese and Chinese fonts, 39.9% of those faces against 52.0% of the others (odds ratio 0.61, interval 0.5 to 0.75). The container's full font set gave the same direction (34.6% against 46.0%).

A face “swings” when its width in a Japanese-capable monospace font (WenQuanYi Zen Hei Mono) differs from its width in DejaVu Sans Mono by 10% or more, both at 24 pixels; the definition is in the statistics script. We do not know why P5 failed. A look we did not plan shows faces with wide or fullwidth characters change width *less* (180 of 710, 25%, against 441 of 906, 49%), which suggests the measure mostly reflects how the two monospace fonts size ordinary Latin letters, not the East Asian Width property. Treat that as a guess. We corrected the two p-values behind our conclusions (P2 and P5) for testing several things at once with the Holm method; the odds ratios and intervals are not adjusted.

LIMITS

Limitations

- **Linux and one browser.** Every rendering result is headless Chromium 141 in a Linux container, with font sets we chose. It does not tell you the share of readers who see a box. Windows, macOS, iOS and Android ship different fonts, and we did not test them.
- **One site's faces.** The faces are the ones we publish. They lean towards cute and chat-style faces and are not a sample of kaomoji at large. The rule for finding face-shaped tiles outside our collections is structural; it misses faces with no bracket pair and keeps two scene-like strings that are not single faces.
- **No full-font result.** The container with every font could not be validated, so we cannot say how many faces break on an ordinary desktop.
- **Standard Markdown only.** Three engines that follow the specification. Chat apps and forums add their own rules, and the Markdown test had no prediction written beforehand.
- **Nothing about people.** We did not ask anyone how they read a face, whether a face with a missing character is still recognisable, or how screen readers speak these characters. No results on those questions exist yet.
- **A model-assisted review.** Separate reviewing models, not people, recomputed the main counts with their own code and rebuilt the published scripts. The check that rendered pages match our list ran in the same browser.
- **The width test did not isolate what it meant to.** See the predictions table.

METHOD

How the work was done

The list of faces. A script reads the site's source files, collects every copy tile (a button whose text is what gets copied) on collection pages, in kaomoji or emoticon sections and anywhere a tile is face-shaped, and adds the generator's starter faces and the decoder's named faces, which it reads by running their data files, not by pattern matching. Pure-emoji tiles are logged and left out. Distinct faces are counted byte for byte. The commit used starts 761bda3b9426 (the full hash is in summary-statistics.json); a rebuild on a later commit gave identical files. On each of the 404 site pages that carry kaomoji tiles, a browser check confirmed that what the page shows equals the list; that check script is not published.

Unicode properties. Script, Script_Extensions, block, East Asian Width, General_Category, grapheme break and emoji properties come from the Unicode 18.0.0 data files listed in source-files.csv with their SHA-256 hashes. Visible units follow the grapheme rules in UAX #29, which we implemented ourselves and checked against the browser's own segmenter: they agreed on all 1,616 faces.

Fonts. For each font set a fontconfig file restricted Chromium to its fonts. We checked every character two ways: by drawing it in the browser and comparing it with an unassigned character, and by reading each font file's character table. Characters that Unicode marks as invisible are not required to have a glyph. Controls were the plain text `abc` and `(^_^)`, which must never be flagged, and unassigned code points, which must always be.

Copy and paste and Markdown. Chromium 141.0.7390.37 driven by Playwright 1.56.1; Markdown engines at the versions named above. A face counts as altered when the visible text differs from the input or when the engine emits any tag.

Statistics. Fisher exact tests and odds ratios with a half added to every cell, intervals by the Woolf method, one logistic regression on the number of other scripts, and Holm adjustment across the two tests that produced the p-values behind our conclusions (P2 and P5). The faces are the whole list, so the intervals describe how firm the comparison is, not a sample.

You can rebuild the list, the font results and the Markdown results with the scripts in the data folder, and recompute the figures on this page from the CSV files with `compute-statistics.py`. `REPRODUCE.md` gives the commands.

RECORD

Corrections

Version 1.0, October 16, 2026. Before publication an independent review by a model, not a person, found that our first list of 1,597 faces missed face-shaped tiles outside the collection pages and counted 11 pictures as faces. We widened the rule, removed the pictures, rebuilt the list (now 1,616 faces) and reran every test. No conclusion changed. We also withdrew an early statement that fewer than 1% of faces break in the full-font container, because that set's detector failed its own check. A second review changed how we word several claims and the title. A corrected version keeps this address, gets a new version number and says what changed.

DATA

Data and files

Full study, version 1.0

PDF · 614 KB

The same text as this page, with title, version and canonical URL on every page.

faces.csv

CSV · 149 KB

Start here: every face, with its characters, scripts, font results and Markdown results.

FILE	CONTENTS	SIZE
faces.csv	Start here: all 1,616 faces with their characters, scripts, whether they break in each font set, and whether Markdown changes them	149 KB
characters.csv	All 692 characters with name, script, block, width class, how many faces use each and how many break it in each font set	51 KB
markdown-results.csv	For every face, whether each Markdown engine changed it, in each placement	78 KB
script-distribution.csv	Faces and characters per Unicode script	1 KB
block-distribution.csv	Faces and characters per Unicode block	3 KB
script-pairs.csv	How often two scripts appear in the same face	2 KB
summary-statistics.json	Every figure on this page as machine-readable numbers	11 KB
source-files.csv	Address, size and SHA-256 hash of each Unicode 18.0.0 file used	1 KB
data-dictionary.csv	What every column in the CSV files means	4 KB
predictions-as-written.md	The predictions we wrote down before running the font and Markdown tests, and every way the tests departed from them	6 KB
REPRODUCE.md	Commands that rebuild the list and every result	3 KB
Scripts	compute-statistics.py, dump-kaomoji-data.mjs, font-coverage.py, font-test-browser.mjs, fontconfig-basic-plus-cjk.conf, fontconfig-basic.conf, make-font-test-input.py, markdown-package-lock.json, markdown-package.json, markdown-test.mjs, reproduce-corpus.py, ucd_reader.py	48 KB

CITE

How to cite this study

UltraTextGen Research. "What Kaomoji Are Made Of, and Where They Break in Basic Fonts and Markdown" Version 1.0, October 16, 2026. <https://ultratextgen.com/research/kaomoji->

unicode-rendering-study/

Please cite the version number and say that the font results are from Linux fonts and that the faces are those on our collection pages.

SOURCES

External sources

The scripts, blocks, character names, categories, widths and break classes counted here come from the Unicode Character Database 18.0.0 (Scripts.txt (<https://www.unicode.org/Public/18.0.0/ucd/Scripts.txt>), ScriptExtensions.txt (<https://www.unicode.org/Public/18.0.0/ucd/ScriptExtensions.txt>), Blocks.txt (<https://www.unicode.org/Public/18.0.0/ucd/Blocks.txt>), EastAsianWidth.txt (<https://www.unicode.org/Public/18.0.0/ucd/EastAsianWidth.txt>) and UnicodeData.txt (<https://www.unicode.org/Public/18.0.0/ucd/UnicodeData.txt>)), with the file list and hashes in source-files.csv (<https://ultratextgen.com/research/kaomoji-unicode-rendering-study/data/source-files.csv>).

That East Asian Width can be Ambiguous and so depend on the context comes from UAX #11 (<https://www.unicode.org/reports/tr11/>); the rules for visible units (grapheme clusters) are in UAX #29 (<https://www.unicode.org/reports/tr29/>); NFC and NFKC are defined in UAX #15 (<https://www.unicode.org/reports/tr15/>); and the Script and Script_Extensions properties, including why a Common character can be shared by several scripts, are described in UAX #24 (<https://www.unicode.org/reports/tr24/>).

How Markdown treats backslashes, emphasis and block quotes is the CommonMark specification, version 0.31.2 (<https://spec.commonmark.org/0.31.2/>); the engines we ran are markdown-it (<https://github.com/markdown-it/markdown-it>) and marked (<https://github.com/markedsjs/marked>), alongside CommonMark's own reference implementation.

That kaomoji draw on many scripts and use combining marks was noted for a Twitter sample by Bedrick, Beckley, Roark and Sproat, "Robust kaomoji detection in Twitter" (<https://aclanthology.org/W12-2107/>) (2012). That the same emoji can be read differently on different platforms comes from Miller and colleagues, "Blissfully Happy" or "Ready to Fight" (<https://doi.org/10.1609/icwsm.v10i1.14757>) (2016), a study of pictures that this study of characters does not repeat.