AQUA RESEARCH · TECHNICAL REPORT 001

AIが作る「動く画面」を壊れにくくする

チャット内アーティファクトの実行時ガード・自己修復・版管理と、その評価で見つかった落とし穴

Making LLM-Generated Interactive Artifacts Robust in a Production Chat: Runtime Guards, Self-Repair, Versioning, and Pitfalls in Their Evaluation

湯村 信司（Shinji Yumura）　AQUA合同会社　2026年10月11日（同日訂正）

訂正（2026年10月11日） 初版の5.2節では、自動修復が「アナログ時計で、2回目の修復で直った」と書いた。その後の調べで、これは修復が必要な不具合ではなく、筆者らの見張りの誤りだったとわかった。見張りは「文字・画像・canvas などがひとつもない画面」を真っ白と判定していたため、色のついた箱だけで描かれた時計を真っ白と取り違えていた。判定は、色・枠・背景のある箱も「見えている」と数えるように直した（YUA への反映は、開発モードの改善とあわせて行う）。動作成功率など、ほかの数字は変わらない。詳しくは Technical Report 002 の6章を参照。

89.6 → 97.9 %
動作成功率（画面が出て、エラーがなく、操作しても壊れない）

6.3 → 0%
実行時エラーが出た割合

±0
審査役が見た「出来の良さ」は変わらなかった（45.8% → 45.8%）

ABSTRACT

Chat assistants increasingly answer requests such as “make me a calculator” with *artifacts*: small HTML/React applications rendered live inside the conversation. In production, these artifacts fail in mundane ways—blocked dialogs in sandboxed frames, missing libraries, malformed code fences—that model quality alone does not address. We describe five changes to the artifact runtime of YUA, a Japanese chat assistant in production: pinned library injection, an in-frame error watcher with bounded automatic

self-repair, name-based versioning, export, and mobile layout. On a 24-task benchmark (48 generations per condition, run concurrently against the pre- and post-change builds), the share of artifacts that render, raise no runtime error, and survive a first interaction rose from 89.6% to 97.9%, and runtime errors fell from 6.3% to 0%. A model judge, however, found no change in overall quality (45.8% vs. 45.8%). Most of the gain came from prevention rather than repair: self-repair fired in only 1 of 48 runs, and that single activation was later found to be a false positive of our blank-screen detector (see Correction). We also report evaluation pitfalls that moved judged pass rates by up to 19 points—more than any effect we measured—including judges unaware of the runtime environment, truncated screenshots, capturing frames mid-repair, and judge sampling variance.

要旨

「電卓を作って」のような頼みに、会話の中でそのまま動く小さなアプリ（アーティファクト）で答えるチャットAIが増えている。本番の画面では、これらは地味な理由で壊れる。安全のための枠の中で `alert()` が無視される、部品が読み込まれない、コードの囲み方を少し間違える、といったものである。これはAIの性能を上げるだけでは防げない。本稿では、筆者らが運営する日本語のチャットAI「YUA」の画面の側に、5つの変更を加えた。部品の自動読み込み、画面内のエラー検知と回数を限った自動修復、名前による版管理、保存とコピー、スマホ対応である。24題×2回のお題で、変更前と変更後を同時に測った。その結果、「画面が出て、エラーがなく、操作しても壊れない」割合は 89.6% から 97.9% に、実行時エラーは 6.3% から 0% になった。一方で、審査役のAIが見た全体の出来は変わらなかった（45.8% と 45.8%）。改善の多くは「直す」より「防ぐ」から来ており、自動修復が働いたのは 48 回中 1 回だけだった。しかもその1回は、あとで筆者らの「真っ白」の判定の誤りだったとわかった（訂正を参照）。あわせて、審査の合格率を最大 19 ポイント動かした評価の落とし穴も報告する。たとえば、動く環境を知らない審査役、途中で切れた画面の写真、修復中の撮影、審査のばらつきなどである。この差は、本稿で測ったどの効果よりも大きかった。

1 はじめに

チャットAIの答えは、文章から「動くもの」へ広がりつつある。グラフ、計算ツール、小さなゲーム、ページの見本などを、AIがその場でコードとして書き、会話の中の安全な枠（`iframe`）で動かして見せる。こ

の仕組みは一般に「アーティファクト」と呼ばれる。

研究の場では、AIが書いたコードの良し悪しは、主に「正しいコードを書けるか」で測られてきた。しかし実際のサービスで利用者が目にするのは、コードではなく**画面**である。どれほどよいコードでも、動かす側の決まりと合わなければ、利用者には「壊れた画面」に見える。たとえば、安全のための枠が確認ダイアログを止めてしまう、決まった配信元以外の部品を読めない、といった場合である。

本稿は、AIそのものには手を入れず、**画面を動かす側**を変えることで、どこまで壊れにくくできるかを報告する。貢献は次の3つである。

1. 本番で動くチャットAIに加えた5つの変更（部品の自動読み込み、エラー検知と自動修復、版管理、保存とコピー、スマホ対応）の設計
2. 変更前と変更後を**同時に**、同じ道具で測るためのお題（24題）と手順、およびその結果
3. 評価の途中で見つかった6つの落とし穴と、それが結論をどれだけ揺らすかの実測

---

2 対象のシステム

YUA は、AQUA合同会社が運営する日本語のチャットAIである。答えの文章は `gemini-3.1-flash-lite` が作る。アーティファクトは、答えの中の決まった印のついたコードの囲み（```` ```artifact:html ```` など）として書かれ、画面がそれを見つけて枠の中で動かす。

安全のため、枠には次の制限をかけている。この制限は変更前も変更後も同じである。

- `sandbox="allow-scripts"`：スクリプトは動くが、親の画面・保存領域・ダイアログ（`alert` など）には触れない
- CSP（読み込みの決まり）：外への通信は禁止（`connect-src 'none'`）、画像は埋め込みのものだけ（`img-src data: blob:`）、スクリプトは決まった配信元（jsDelivr）だけ

外の画像を禁じているのは、画像の住所に会話の中身を載せて外へ送る、という抜け道を断つためである。この強い制限があるために、ふつうのウェブページなら動くコードでも枠の中では動かないことがある。これが本稿の出発点である。

---

3 提案する仕組み

3.1 部品の自動読み込み（グラフ・3D・アニメーション）

使ってよい部品を4つに絞り、版を固定した。Chart.js 4.5.1、three.js 0.186.1、D3 7.9.0、GSAP 3.15.0 である。AIには配信元のURLを書かせない。画面の側がコードの中の名前（`new Chart(`、`THREE.`、

`d3.select`、`gsap.to` など）を見つけ、決まった版を差し込む。`import * as THREE from 'three'` のような今の書き方にも対応するため、読み込み先の表（importmap）を入れた。三次元の操作部品（`three/addons/…`）もこれで使える。AIが自分で書いた読み込みの行は外す。URLの書き間違いや偽物の部品を防ぐためである。

React の画面では、AIの書いた `import React from 'react'` を、先に読み込んだ React にそのままつなぐ小さな橋渡し（`data:` の住所で作る）を用意した。React が二重に読み込まれると `useState` などが動かなくなる。この橋渡しで、それを避けている。

3.2 ダイアログの置き換え

枠の中では `alert()`・`confirm()`・`prompt()` は止められ、何も表示されない。AIはこれらを結果の表示や「消しますか？」の確認によく使う。そこで、画面の中にお知らせを出す同名の関数に置き換えた。AIへの指示でも使わないように頼んでいるが、守られないことがあるため、画面の側でも受け止める。

3.3 エラーの検知と自動修復

枠の中に、見張りの小さなスクリプトを**最初に**差し込む。拾うのは次の4つである。

- 例外（部品の読み込み失敗を含む）
- 処理されなかった Promise の失敗
- `console.error`
- 2.5秒たっても何も表示されない「真っ白」

拾ったものは親の画面に伝える。当初は見張りをページの最後に置いていたが、それでは読み込みの途中で起きるエラーを取りこぼしていた。

エラーが届くと、1.2秒待ってエラーをまとめる。そのうえで、元のコード・エラー・利用者の頼みごとを修復用の窓口に送り、直したコード全体を受け取る。受け取ったコードは、会話の中のそのコードと差し替え、保存する。無限に直し続けないための決まりは次の通りである。

- 同じ画面について自動修復は2回まで
- 直すのは、この画面を開いてから作られた答えだけ（昔の会話を開いただけでは直さない）
- 修復の窓口には、1分あたり・1日あたりの回数制限をかける

自動で直せなかったときは「直す」ボタンを出す。

3.4 版の管理

AIに、画面に短い名前を付けさせる（```` ```artifact:html:売上グラフ ````）。前の画面を直すときは同じ名前で出させる。画面の側は、会話全体から同じ名前のものを順に並べ、「版 1/2」「版 2/2」のように切り替えて

見せる。前の版に戻すこともできる。

3.5 AIへの指示

画面の側の変更に合わせて、AIへの指示も改めた。見た目で示すべき頼みには必ずアーティファクトで答えること。使える部品はこの4つで、読み込みの行は書かないこと。グラフには軸の目盛り・単位・凡例を付けること。ダイアログは使わないこと。スマホの幅（360px）でも崩れないこと。

3.6 保存・コピー・スマホ対応

画面の下に「コピー」「保存」「全画面」の帯を置いた。保存は、部品の読み込みも含めた1つのHTMLファイルとして端末に書き出す。

高さの合わせ方も改めた。中身が枠より長いときは全体の長さに合わせる。枠に収まるときは、中身の下の端までに縮める。画面いっぱいに描く作品（`height: 100%` や `100vh`）は中身が枠に合わせて伸び縮みするため、最初の高さ（420px）のまま見せる。以前は、こうした作品が細い帯に潰れていた。

---

4 評価の方法

4.1 お題

利用者が実際に頼みそうなものを24題用意した。グラフ5題、3D 2題、その他17題（計算ツール、ゲーム、表、アニメーション、ページの見本など）である。全文は付録に載せた。それぞれ2回ずつ頼み、1つの条件につき48回となる。

4.2 自動の判定

本物のチャット画面（Chrome、1280×900）で、ふつうの利用者と同じように頼む。会話を残さない設定にして送り、出てきた画面を次の4点で調べる。4つすべてを満たしたものを「動作成功」とした。

- **画面が出た**：答えの中にアーティファクトがある
- **エラーなし**：枠の中で例外・`console.error`・エラー表示が出ない
- **真っ白でない**：写真の色のばらつき、または文字の量がしきい値を超える
- **押しても平気**：画面の中の最初のボタンを1回押しても、新しいエラーが出ない

4.3 審査役による採点

出来の良さは、別のAI（`gemini-3.8-flash`）に採点させた。渡したのは、頼みごと・画面の写真・コードの3つである。2点は頼んだ通り、1点は一部欠けや見た目の崩れ、0点は動かないか別物である。審査役には、この画面では部品が自動で読み込まれること、写真の下が切れているのは撮影の都合であることを伝えた。これは6章の落とし穴を避けるためである。同じ画面を3回採点させ（1回目は温度0、2・3回目は0.7）、真ん中の点を使った。

4.4 同時に測る

AIの答えは日や時間で変わりうる。そのため、変更前のコード（本番の1つ前の版）を別の作業フォルダに取り出し、別の番号で立ち上げた。そして変更後と**同じ時刻に、同じ測る道具で**測った。審査も、両方を同じ審査役が最初から採点した。

---

5 結果

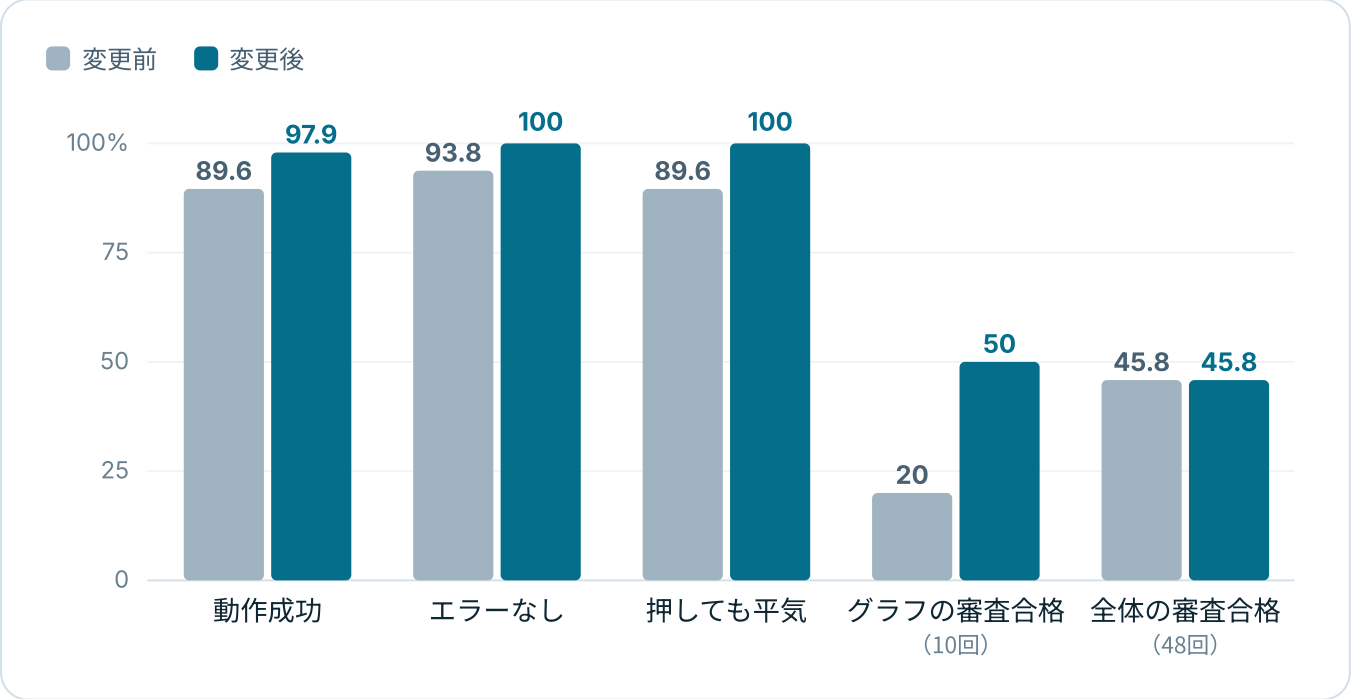

図1　変更前と変更後（各48回、同時に測定）。壊れにくさの指標は上がったが、審査役が見た全体の出来は変わらない。

指標	変更前	変更後	p値
画面が出た	47/48（97.9%）	48/48（100%）	1.00
エラーなし	45/48（93.8%）	**48/48（100%）**	0.24
真っ白でない	47/48（97.9%）	47/48（97.9%）	1.00
押しても平気	43/48（89.6%）	**48/48（100%）**	0.056
動作成功（上の4つすべて）	**43/48（89.6%）** **95%区間 77.8–95.5**	**47/48（97.9%）** **95%区間 89.1–99.6**	**0.20**
審査で2点（全体）	22/48（45.8%）	22/48（45.8%）	1.00
審査で2点（グラフ）	2/10（20%）	5/10（50%）	0.35
審査で0点	1/48（2.1%）	0/48（0%）	1.00
審査の平均点（0〜2）	1.44	1.46	—
答えが出るまでの時間（中央値）	26秒	27秒	—

表1　主な結果。p値はフィッシャーの正確確率検定（両側）、区間はウィルソンの方法による。

動作成功は 43/48 から 47/48 に、実行時エラーは 3 件から 0 件に減った。ただし回数が少ないため、どの差も統計的にはっきりしたとは言えない（「押しても平気」の p = 0.056 が最も近い）。お題ごとに見ると、2回のうち動いた回数が増えたお題が4つ、減ったお題が1つあった。

5.1 変更前に何が壊れていたか

変更前の失敗5件のうち**4件は、ダイアログが止められたこと**によるものだった。内訳は、BMIの計算結果を `alert` で出したもの、お絵かきボードの「全部消しますか？」を `confirm` で聞いたもの、スネークゲームのゲームオーバーを `alert` で出したもの（2件）である。残る1件は、住宅ローンの試算でアーティファクトが出なかった。答えのコードを調べると、`alert` か `confirm` を使っていたのは変更前で48回中6回あった。変更後はAIへの指示で禁止したが、それでも4回あった。

変更後の失敗1件は、マウスを動かすまで何も描かれないパーティクルのアニメーションである。止まった状態の写真では真っ白に見えるため、これは測る道具の限界とみなすのが妥当である。

5.2 自動修復はほとんど働かなかった

変更後の48回のうち、自動修復が働いたのは1回だけだった（アナログ時計）。**ただしこれは誤った作動だった**（下の訂正を参照）。時計は文字のない色の箱（div の文字盤と針）だけで描かれており、見張りが「真っ白」と誤って判定した。正しく動いていた画面を、2回書き直したことになる。仕組みそのものが正しく動くことは、わざと壊したコード（定義されていない関数を呼ぶ）で確かめている。この場合は1回の修復で直り、「直しています」の表示が出て、直したあとにエラーは出ず、ボタンも押せた。つまり今回の改善の大部分は、**壊れてから直すことではなく、壊れないようにする工夫**（ダイアログの置き換え、部品の自動読み込み）から来ている。

5.3 途中で見つかった新しい失敗

開発の途中で一度測ったとき、48回中3回（6.3%）でアーティファクトがまったく出なかった。3.4節の「名前を付ける」書き方を入れた結果、AIが印の一部を省いて ```` ```react:お絵かきボード ```` と書くようになったためである。改善のために入れた決まりが、別の壊れ方を生んでいた。印が欠けていても名前があれば画面として扱うように直した。あわせてAIへの指示にも注意を足し、該当の3題を測り直すと6回中6回で画面が出た。お題の集まりで毎回測ることが、こうした後戻りを見つける唯一の手段だった。

---

6 評価の落とし穴

審査役のAIを使った評価は手軽だが、設定の細かな違いで結果が大きく動いた。表2は、ほぼ同じ変更を3通りの審査のしかたで測ったものである。

審査のしかた	変更前	変更後	差
A：強い審査役・1回採点・環境を伝えない・古い測る道具	58.3%	47.9%	−10.4
B：Aに「部品は自動で読み込まれる」等を伝える	64.6%	50.0%	−14.6
C：同時に測定・道具を直す・3回採点の中央値（本稿の結果）	**45.8%**	**45.8%**	**0.0**

表2　審査で2点を取った割合。AとBの変更後は、5.3節の「印の欠け」を直す前のもの。強い審査役は `gemini-3.1-pro-preview`、Cの審査役は `gemini-3.8-flash`。同じ変更前のデータでも、審査のしかたで 45.8% から 64.6% まで、19ポイント近く動いた。

P1 審査役が、動く環境を知らない

変更後は、部品の読み込みを画面の側で行う。そのため、コードには読み込みの行がない。何も伝えない審査役は、これを「Chart.js を読み込むタグがないので単体ではエラーになる」と減点した。減

点は48回中6回あり、変更後の成績だけを不当に下げていた。**改善したところが、審査では欠点に見える**という逆転が起きうる。

P2 画面の写真が途中で切れる

縦に長い画面を写すと、ブラウザの窓の外は黒く写る。審査役はこれを「画面の下半分が黒く塗りつぶされている」と減点した（カフェの紹介ページで変更前・変更後とも）。写す前に窓を画面の高さまで広げるよう直した。

P3 修復の途中を撮ってしまう

エラーに気づいてから修復が始まるまでには数秒ある。初めの道具は修復が始まる前に「修復中ではない」と判断し、「直しています」の表示がかかった画面を撮っていた。審査役はそれを「エラーメッセージが被さっている」と減点した。修復していない状態が8秒続くまで待つように直した。

P4 審査のばらつき

同じ画面を3回採点させたとき、3回とも同じ点だったのは96件中69件（71.9%）だった。1回だけの採点では、お題ごとの点は3割近くの確率で揺れる。48回という規模では、この揺れが差の大きさを上回る。

P5 審査役の回数制限

強い審査役は1日に使える回数の上限（250回）に届き、途中から採点に失敗した。審査役が混ざると比べられなくなるため、別の審査役で変更前・変更後の両方を最初から採点し直した。

P6 変更前を、別の時刻・別の道具で測る

最初は、変更前を先に測り、あとから変更後を測っていた。しかし途中で測る道具そのもの（P2・P3）を直したため、前後の差に「道具の差」が混ざった。変更前のコードを別のフォルダで立ち上げ、同じ道具で同時に測り直したものが表1である。

> 審査役のAIによる評価で「改善した」「悪化した」と言う前に、次の4つを確かめる必要がある。審査役が動く環境を知っているか、写真が欠けていないか、比べる両方を同じ道具・同じ

> 時刻で測ったか、点のばらつきが差より小さいか。本稿では、これらを直す前の結論（「出来が10ポイント下がった」）と、直したあとの結論（「変わらない」）が逆だった。

---

7 考察と限界

7.1 「防ぐ」は「直す」より効く

自己修復はよく話題になるが、今回の改善のほとんどは、AIがよく使い、枠の中では止まってしまう書き方を**画面の側で受け止める**ことから来た。AIへの指示で禁止しても、48回中4回は守られなかった。動かす側で受け止めれば、AIが守らなくても壊れない。修復は、防ぎきれなかったものの最後の受け皿と位置づけるのがよい。修復はAIをもう一度呼ぶため、時間とお金もかかる。

7.2 壊れにくさと出来の良さは別のもの

動作成功は上がったが、審査役の見た出来は変わらなかった。今回の変更は「動くかどうか」には効くが、「頼んだ通りの見た目と機能になっているか」には効かない。審査役の指摘も、ほとんどは仕様の不足だった（Enterキーで追加できない、黒鍵の長さが本物と違う、など）。出来を上げるには、作るAIそのものの力や、作る前に設計させる工夫など、別の手当てが要る。

7.3 限界

- **規模が小さい**：24題×2回。どの差も5%水準ではっきりしたとは言えない。今回の結果は「壊れる理由が消えた」という事例の報告として読むのが妥当である。
- **作るAIと審査役が同じ系統**：どちらも Gemini である。同じ系統のAIを甘く見る偏りがありうる。
- **人による評価がない**：出来の良さは審査役のAIだけで測った。
- **操作は最初のボタン1回だけ**：ドラッグ・キー入力・音は自動では確かめていない。
- **変更を1つずつ切り分けていない**：画面の側の変更と同時に、AIへの指示（ダイアログを使わない、部品の使い方、スマホの幅）も変えた。どの変更がどれだけ効いたかを、1つずつ外して測ってはいない。
- **お題は筆者が作った**：実際の利用者の頼みごとの分布とは違いうる。お題も日本語のみ。

---

8 関連研究

AIが自分のコードの誤りを実行結果から直す試みとして、Self-Debugging [1] や Reflexion [2] がある。本稿の自動修復はこれらに近い。ただし、修復の手がかりを画面の中の実行時エラーと「真っ白」から集める点、無限に回さないための回数制限を本番の運用条件として扱う点が異なる。

AIによる採点（LLM-as-a-Judge）の偏りは [3][4] で詳しく調べられている。本稿の落とし穴P1〜P3は、それらとは別の種類の偏りである。審査役に渡す情報（動く環境の説明や写真）が欠けていることから生まれる。画面の見本からウェブページを作らせて評価する研究に Design2Code [5] がある。同じ画面を2回以上作らせて自動判定と審査を組み合わせる点は本稿も同じだが、本稿は本番のチャットの枠の制約を評価に含めている。

9 まとめ

チャットAIが作る「動く画面」は、AIの力とは別の、動かす側の決まりとのすれ違いでよく壊れる。部品の自動読み込み、ダイアログの置き換え、エラー検知と回数を限った自動修復を画面の側に入れると、動作成功は 89.6% から 97.9% に、実行時エラーは 6.3% から 0% になった。一方で、出来の良さは変わらなかった。そして、AIによる審査を使って前後を比べるときは、審査役への情報の渡し方と測り方の違いだけで、結論が逆になりうる。今後は、人による評価、より多くのお題、別の系統の審査役で確かめるとともに、出来の良さそのものを上げる工夫に取り組む。

A 付録：お題の一覧

ID	分類	頼みごと（原文）
calc	基本	電卓を作って。四則演算とクリアができるもの
todo	基本	やることリストを作って。追加・完了チェック・削除ができるもの
bmi	基本	身長と体重を入れるとBMIと判定が出る計算ツールを作って
timer	基本	25分のポモドーロタイマーを作って。開始・一時停止・リセット付き
clock	基本	針が動くアナログ時計を作って
converter	基本	長さ（mm・cm・m・km・インチ・フィート）の単位変換ツールを作って
table	データ	社員10人の名前・部署・年齢の表を作って。列の見出しを押すと並べ替えできて、名前で絞り込みもできるもの
quiz	状態	日本の地理の4択クイズを5問作って。最後に点数が出るもの
flashcard	状態	英単語の暗記カードを作って。押すとくるっと裏返るもの

ID	分類	頼みごと（原文）
kanban	操作	ドラッグで動かせるカンバンボード（未着手・作業中・完了）を作って
drawing	操作	マウスや指で絵が描けるお絵かきボードを作って。色と太さを変えられて、消すボタンも付けて
snake	ゲーム	スネークゲームを作って。矢印キーとスマホのボタンで操作できるもの
typing	ゲーム	ローマ字のタイピング練習ゲームを作って。60秒で何文字打てたか出るもの
particles	動き	マウスを追いかけるきれいなパーティクルのアニメーションを作って
piano	音	1オクターブの鍵盤を押すと音が鳴るピアノを作って
physics	動き	重力でボールが跳ねる物理シミュレーションを作って。クリックでボールを増やせるもの
bar-chart	グラフ	1月〜12月の売上の棒グラフを作って。データは適当で大丈夫
line-chart	グラフ	3つの店舗の月ごとの来客数を比べる折れ線グラフを作って。凡例付きで
pie-chart	グラフ	家計の支出の内訳を円グラフで表示して。項目を押すと金額が出るもの
dashboard	グラフ	売上・客数・平均単価の数字のカードと、月ごとの推移のグラフが並んだダッシュボードを作って
mortgage	グラフ	住宅ローンのシミュレーターを作って。借入額・金利・年数から毎月の返済額と、返済の推移のグラフを出すもの
cube-3d	3D	マウスで回せる3Dの立方体を作って
solar-3d	3D	太陽のまわりを惑星が回る3Dの太陽系を作って
landing	見本	カフェのおしゃれな紹介ページを作って。メニューとアクセスも入れて

測定日：2026年10月10日。作るAI：gemini-3.1-flash-lite。審査役：gemini-3.8-flash（表2のA・Bは gemini-3.1-pro-preview）。ブラウザ：Google Chrome（ヘッドレス、1280×900）。部品の版：Chart.js 4.5.1／three.js 0.186.1／D3 7.9.0／GSAP 3.15.0／React 18.3.1／Babel 7.26.4。

参考文献

1. X. Chen, M. Lin, N. Schärli, D. Zhou. “Teaching Large Language Models to Self-Debug.” *ICLR*, 2024. arXiv:2304.05128.
2. N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, S. Yao. “Reflexion: Language Agents with Verbal Reinforcement Learning.” *NeurIPS*, 2023. arXiv:2303.11366.
3. L. Zheng, W.-L. Chiang, Y. Sheng, et al. “Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena.” *NeurIPS Datasets and Benchmarks*, 2023. arXiv:2306.05685.
4. P. Wang, L. Li, L. Chen, et al. “Large Language Models are not Fair Evaluators.” *ACL*, 2024. arXiv:2305.17926.
5. C. Si, Y. Zhang, R. Li, Z. Yang, R. Liu, D. Yang. “Design2Code: Benchmarking Multimodal Code Generation for Automated Front-End Engineering.” *NAACL*, 2025. arXiv:2403.03163.