# Generating Executable Code in a Visually Opaque Unicode Language: A CLI Case Study

Daniele Scaratti
Independent Researcher
ORCID: 0009-0008-3931-8821
DOI: 10.5281/zenodo.24011466

November 6, 2026

**Abstract.** Can a language model use a supplied specification to produce executable programs in a notation that offers an unaided reader almost no visual cues? We study emd, a stack language written in eight similar Unicode dashes, through command-line clients with tools disabled. Across *49* tasks, we collected *1150* Claude responses and an incomplete extension of *573* OpenAI responses. Exact copying is often successful, and in a complete Sol basic-prompt sample, *70* of *72* program answers produce the required behavior. Correct content does not always arrive in usable source form: output formatting rejects *33.3%* of Haiku's correct answers under the basic prompt, reduced to *0.0%* when the consumer's requirements are stated explicitly. We report exact-answer and normalized-content scores separately, together with token-cost estimates. These observations characterize the tested model–CLI–prompt configurations. They do not measure human readability, isolate the effect of visual opacity, or establish that the language was absent from training data.

## 1. Introduction

A language model can return source code whose rendered characters give a human reader almost no indication of its structure. This observation invites a precise question before any broader claim about machine-written code: can the model actually produce programs that execute as requested? We study that question using emd, a stack language whose source consists of *eight* visually similar Unicode dashes. The language and its verification are described separately [6]; this article concerns responses produced from its supplied specification.

The representation is visually opaque, but its semantics is explicit. Code-point inspection or readable assembly recovers the distinctions that a line of glyphs obscures. A text model need not encounter those distinctions as a human sees them. Consequently, successful generation would demonstrate use of the specified representation, not superior visual perception or an ability beyond human understanding.

We examine three observable behaviors: exact reproduction of the symbols, construction of programs that satisfy task tests, and adherence to the input format required by the compiler. The last distinction matters because a correct program enclosed in Markdown is not an accepted emd source. We therefore score each reply as received and again after a defined normalization, and measure the estimated token cost per immediately usable answer.

The study contains *1150* responses from two requested Claude models and *573* responses from two requested OpenAI models, all reached through command-line clients with tools disabled. It provides a case study of executable-code generation and its output interface. The collected samples do not establish the effect of visual opacity itself, since there is no matched readable-representation control or human-reader study.

## 2. Related Work

HumanEval evaluates generated programs through functional tests [1]. We also use executable behavior as an outcome, while distinguishing failure of the task from rejection of the answer's presentation. Neither a text that resembles source nor a program that merely compiles is sufficient to count as a successful program answer.

LatentMD separates content from Markdown boundary errors in its setting [2]. Here the consumer is a non-Markdown decoder: wrappers that assist conversational presentation can invalidate its input. emd makes this boundary unusually small and explicit. Our measurements concern that operational contract and the tested client configurations, rather than an explanation of why models produce a particular presentation style.

## 3. Method

### 3.1 Language and tasks

emd has *32* instructions encoded by pairs drawn from U+002D, U+2010, U+2011, U+2012, U+2013, U+2014, U+2015 and U+2212. Its compiler accepts only these code points and newlines, with at most *32* code points on a line. Spaces, backticks and visually similar characters outside the alphabet are errors. Line breaks carry no execution meaning, but overlong lines are rejected.

The task suite contains *87* tasks. We select *49*: all *31* hard tasks and the first *three* easy tasks of each kind. Table 1 describes the six kinds. The hard tier uses longer sources, and some program tasks consume input not shown in the prompt. Thus producing a fixed output string is not sufficient for all program tasks. Tier and task kind also differ in other respects; they are not controlled measures of a single reasoning demand.

Table 1: Task families. The first three require executable source; the others have textual reference answers.

| Kind | Required answer |
|---|---|
| generate | program with specified behavior |
| complete | continuation of a supplied program |
| repair | correction of one wrong code point |
| explain | instruction listing |
| distinguish | index where two strings differ |
| copy | exact reproduction of supplied dashes |

### 3.2 Clients, conditions and collection

The primary collection requests `claude-haiku-5-5` and `claude-sonnet-5-5` through the Claude Code CLI. An additional collection requests `gpt-6-luna` and `gpt-6.1-sol` through Codex CLI `0.166.0`, at medium reasoning effort. That setting does not establish a compute budget comparable to Claude's. Names identify requested configurations; the Codex records contain no served model identifier, and three Sonnet calls also record usage of another model name.

Each task and repetition is a separate call in an isolated directory. The language specification is supplied in the system context; the OpenAI collection reuses the saved specification from the Claude experiment. Personal configuration, project instructions and tools are disabled. The models receive no compiler or scorer feedback, and completed replies are retained without resubmission, regardless of score. Adaptation here means using the specification in the current call, not learning across calls.

Each requested model is assigned *three* repetitions per task under four conditions:

- **bare**: the task as written, already ending with an instruction such as "Answer with the source only."
- **explicit**: adds that the entire reply goes unchanged to a strict decoder and must omit Markdown, fences, backticks, bold and extra spaces.
- **schema**: the answer is returned in a JSON field, separating it from surrounding presentation text.
- **plain**: the *bare* task with a system specification stripped of Markdown.

The explicit wording excludes newlines as well for copying tasks, whose target is a flat string. Program answers retain their permitted newlines. The schema condition constrains the response channel, not every rule of the language inside the field.

A complete model–condition cell contains *147* answers. The Claude collection has *1150*: Sonnet *plain* contains *124*, and three Haiku conditions contain *146* each. Collection stopped before any Opus call. The OpenAI extension contains *573* completed answers and ended early after repeated CLI timeouts; pending calls were cancelled. Sol *bare* and *explicit* are complete. Other collected OpenAI cells are partial, and no OpenAI *plain* responses were collected. Tables 2 and 3 give the denominators. Missing replies are not scored as incorrect, and incomplete task mixtures are not treated as interchangeable samples.

### 3.3 Scoring

Program answers go through the project's real compiler and VM, with a common bound of *one million* execution steps. They must compile, halt within that bound and produce the required output on the task's recorded input. For *complete*, the answer follows the supplied prefix on a new line. Correctness here means satisfying these task tests, not a proof of behavior on every possible input.

Other task kinds compare with a reference string, index or instruction listing. Their scorer strips surrounding whitespace; instruction listings additionally ignore whitespace around each line. Therefore "strict" denotes the task-specific consumer contract: exact source bytes for program answers, and these stated tolerances for textual answers.

Each answer is classified as *clean* if it passes that scorer as received; *wrapped* if it passes only after removing fences, backticks, bold or spaces; *layout* if it requires that normalization and line relayout; or *wrong* if it remains incorrect. Relayout applies only to program answers. A wrong wrapped answer remains *wrong*, so the analysis does not count every presentation violation as recoverable content.

*Compiles* is the fraction of program answers accepted by the compiler, regardless of their output. *Strict* is the fraction of all answers classified as clean. *Content* is the fraction correct after the specified normalization. Format loss is conditional on recoverable correctness:

$$L = \frac{wrapped + layout}{clean + wrapped + layout}.$$

It is not the fraction of all replies that contain Markdown. The strict–content difference is instead a percentage-point change over all answers.

### 3.4 Statistical comparisons

Prompt comparisons match task identifiers and repetition indices. Those indices identify separate calls, not shared random seeds. To avoid treating repetitions as independent task-level evidence, our main inference counts each task once, comparing its successes on repetitions available in both conditions. Better and worse task totals are assessed with a sign test. Answer-level exact McNemar comparisons [4] are supplied as supplementary descriptions and do not account for that dependence.

The Wilson intervals [3] in Table 2 treat answers as independent and are descriptive approximations, not task-cluster intervals. All comparisons are exploratory, with no adjustment for multiple testing. We do not perform complete-design tests on partially collected OpenAI conditions.

## 4. Results

### 4.1 Copying symbols and constructing programs

Exact copying is often successful under *schema*: Sonnet reproduces all *9* of *9* sampled strings of *1024* code points, and Haiku *8* of *9*. The copying data contain *1* single-code-point substitution. These observations establish fidelity

on the sampled strings; they do not explain how the model represents the symbols.

Program generation is also possible without tool feedback. In the complete Sol *bare* sample, all *72* program answers compile and *70* satisfy their behavior tests. Under *explicit*, all *72* program answers satisfy those tests. This supplies direct successful instances of executable code in the specified representation.

Performance is not uniform across tasks. Haiku and Sonnet's *bare* content scores over all task kinds are *71.4%* and *87.1%*, respectively. On hard *complete* tasks, their pooled *bare* content score is *45.8%*. The task breakdown appears in Appendix A. These are observed differences between task families, not an isolation of reasoning difficulty or visual confusability.

### 4.2 Correct content and rejected presentation

A real answer illustrates why executable behavior and format adherence need separate measurements. Asked for a program that writes `A`, with "Answer with the source only," Haiku returned:

```
```
----------
```
```

The source inside the fence produces the requested output, but the exact reply is rejected at the first backtick. Asked for an index, the same model also returned `**21**`: a correct value with a wrapper the target consumer does not accept.

For Haiku under *bare*, *47.6%* of answers pass strictly and *71.4%* pass after normalization. Thus *33.3%* of its recoverably correct answers are lost to presentation. Sonnet's corresponding loss is *6.2%*. Figure 1 shows strict and content outcomes across conditions; Table 2 gives the rates and descriptive intervals. Content errors remain distinct from this recoverable formatting loss.

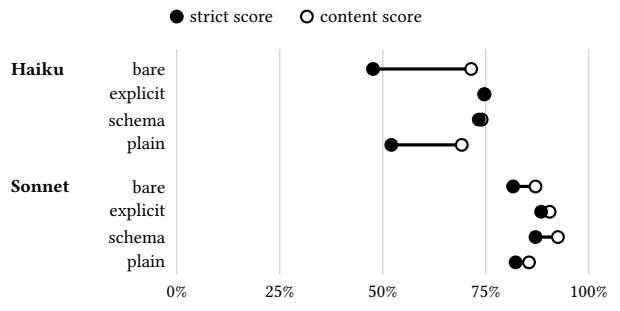



Figure 1: Claude strict and content success rates. The segment between the two marks is the percentage-point gain after wrapper removal and line relayout.

### 4.3 Response to additional instructions

Under *explicit*, Haiku's compiling-program rate rises from *29.2%* to *71.8%*, and its observed format loss falls to *0.0%*. The task-level strict comparison favors the warning on *24* tasks and the basic prompt on *4* ($p < 0.001$). Sonnet has a smaller initial loss; its task-level comparison is *10* better and *5* worse under the warning ($p = 0.30$), providing no detectable improvement in this sample.

A structured channel is not equivalent to a complete language validator. Haiku's observed loss under *schema* is *0.9%*; that and the zero observed under *explicit* do not establish equivalence of the interventions. Sonnet reaches *92.5%* content and *87.1%* strict success under *schema*; the difference comes from overlong lines. An absence of observed wrapper loss is likewise not a guarantee on future calls.

Removing Markdown from the system specification does not produce a detectable strict improvement. Haiku reaches *52.1%* under *plain*, against *47.6%* under *bare*; the task-level sign test gives $p = 0.54$. This does not identify whether Markdown in the context contributes to the behavior. Supplementary paired counts are given in Appendix A.

### 4.4 The incomplete OpenAI extension

Table 3 includes every completed OpenAI answer, with sample sizes and program denominators. Sol's complete *bare* cell has *145* strict successes out of *147* across task kinds (*98.6%*); its complete *explicit* cell has *147*. The difference is confined to *one* task, with task-level sign-test $p = 1.00$. These data do not establish an improvement from the warning.

The partial Luna cells and Sol *schema* are reported descriptively. Their task coverage is incomplete and uneven. Percentages do not make those mixtures equivalent to complete cells, and the client differences preclude attributing an observed provider gap to model weights alone. The rows document successful and unsuccessful responses under their recorded conditions, not a general ranking of models.

## 5. Token Costs

For any cell with strict successes, estimated cost per immediately usable answer is the recorded or estimated token cost divided by its number of strict successes. It includes the usage of completed incorrect replies in the numerator. A correct reply requiring wrapper removal is not an immediately usable answer under this definition.

The Claude client reports US$*13.52* in token cost at its stated rates. Under *bare*, Haiku's estimated cost per usable answer is US$*0.006*, against US$*0.004* under the warning. The intervention therefore improves observed usable output and reduces that estimate in this sample. Sonnet remains near US$*0.023*. These values price recorded usage and do not allocate subscription fees.

Codex reports tokens but no dollar charge per request. Table 3 therefore uses a separate, illustrative API-equivalent estimate at published Standard short-context rates [5]. Per million input, cached-input and output tokens respectively, the rates are US$*0.10*, US$*0.01* and US$*0.50* for Luna, and US$*2.00*, US$*0.10* and US$*10.00* for Sol. With reported input $I$, its cached subset $C$, and re-

Table 2: Claude responses by requested model and condition, in percent. Strict and Content include approximate 95% Wilson intervals; see Section 3.3. Compiles: program tasks whose exact answer the compiler accepts. Strict: passes the task-specific scorer. Content: passes after wrapper removal and line relayout. Loss: format loss. Cost: US dollars for the row, at the prices the client reports. Per usable: cost of one strict success. 147 answers per row, except 146 for Haiku under *explicit*, *schema* and *plain*, and 124 for Sonnet under *plain*.

| Model | Condition | Compiles | Strict | Content | Loss | Cost | Per usable |
|---|---|---|---|---|---|---|---|
| Haiku | bare | 29.2 | 47.6 (39.7–55.7) | 71.4 (63.7–78.1) | 33.3 | 0.41 | 0.006 |
| | explicit | 71.8 | 74.7 (67.0–81.0) | 74.7 (67.0–81.0) | 0.0 | 0.42 | 0.004 |
| | schema | 67.6 | 73.3 (65.6–79.8) | 74.0 (66.3–80.4) | 0.9 | 0.48 | 0.004 |
| | plain | 33.8 | 52.1 (44.0–60.0) | 69.2 (61.3–76.1) | 24.8 | 0.46 | 0.006 |
| Sonnet | bare | 76.4 | 81.6 (74.6–87.1) | 87.1 (80.7–91.6) | 6.2 | 2.76 | 0.023 |
| | explicit | 80.6 | 88.4 (82.3–92.7) | 90.5 (84.6–94.2) | 2.3 | 3.03 | 0.023 |
| | schema | 80.6 | 87.1 (80.7–91.6) | 92.5 (87.1–95.8) | 5.9 | 3.74 | 0.029 |
| | plain | 75.4 | 82.3 (74.6–88.0) | 85.5 (78.2–90.6) | 3.8 | 2.22 | 0.022 |

Table 3: Codex CLI sample. Each cell planned 147 answers; only collected cells are shown. Compiles gives the percentage and count of collected program answers accepted by the compiler. Strict and Content are percentages over all $n$ answers. API equiv.: estimated standard API token cost of completed calls, in US dollars, not subscription expenditure. Per usable divides that estimate by strict successes; a dash denotes no strict success.

| Requested model | Condition | $n$ | Compiles | Strict | Content | API equiv. | Per usable |
|---|---|---|---|---|---|---|---|
| `gpt-6-luna` | bare | 67 | 21.2 (7/33) | 26.9 | 26.9 | 0.0557 | 0.0031 |
| | explicit | 16 | 42.9 (3/7) | 18.8 | 18.8 | 0.0120 | 0.0040 |
| | schema | 80 | 30.4 (14/46) | 36.2 | 37.5 | 0.0770 | 0.0027 |
| `gpt-6.1-sol` | bare | 147 | 100.0 (72/72) | 98.6 | 98.6 | 2.4459 | 0.0169 |
| | explicit | 147 | 100.0 (72/72) | 100.0 | 100.0 | 1.6561 | 0.0113 |
| | schema | 116 | 100.0 (57/57) | 100.0 | 100.0 | 1.6990 | 0.0146 |

ported output $O$, the estimate is

$$E = \frac{(I - C)r_i + Cr_c + Or_o}{10^6}.$$

No cache writes are reported. Separately reported reasoning tokens are not added to the output total again. The estimate for completed OpenAI calls is US$*5.95*. It is not an invoice or a total experimental expenditure: timed-out and cancelled calls have no recorded token usage, and subscription charges cannot be assigned from these data. The two clients' dollar columns should consequently retain their different cost-accounting labels.

## 6. Discussion

The successful program answers show that a supplied specification can support executable-code generation in this representation. The source is difficult to distinguish by ordinary glyph appearance, but that does not make its semantics inaccessible. Neither these successes nor the copying results establish that the model sees the glyphs as a person would, or that the task was absent from training.

The strict–content difference concerns a separate interface property. Markdown can be appropriate in a conversation while invalidating a compiler input. A wrapper failure is part of performance when the target is immediate consumption; it is recoverable presentation loss when the target is content independent of that interface. The evaluation target must be stated before interpreting either score. For the paired Haiku–Sonnet data, the strict score shows a larger observed gap than the normalized-content score (Appendix A); both still favor Sonnet in those conditions.

These observations make emd a useful case study rather than a general test of AI competence. Making the consumer explicit helps some tested configurations, but does not replace validation of the reply. A structured field separates content from surrounding text while leaving the language's own grammar and layout rules to be checked. The data support that practical distinction without identifying its training-level cause.

## 7. Limitations

**Representation and scope.** There is no matched experiment in readable assembly and no human readability study. Visual opacity describes the rendered source; it is not an experimentally isolated cause of model performance. The results cover one small language, a selected task set and a small number of requested configurations.

**Collection and missingness.** The OpenAI collection stopped early, and the Claude collection also has incomplete cells. Missingness may depend on task and model–client behavior; the partial samples are not assumed representative. Task difficulty labels, source length and required behavior are not independently varied.

**Clients and sampling.** All calls use CLIs, not direct API requests. Claude Code adds a preamble that was not inspected; Codex supplies the saved specification as its base instructions. Temperature and random seeds were not controlled, and calls were collected in condition batches. Client behavior, context handling and temporal variation remain possible contributors. Requested names do not identify immutable served snapshots.

**Scoring and inference.** Normalization defines what

counts as recoverable formatting; another repair policy could classify some answers differently. Functional tests on recorded inputs do not prove program equivalence for all inputs, and scorer or implementation defects remain possible. Repetitions are dependent, Wilson intervals are approximate for this design, and the comparisons are exploratory. These limits prevent interpreting the results as a general model ranking.

**Stability.** An earlier retained run, on a language version with *29* instructions and a different line-handling scorer, measured Sonnet *bare* loss at *17.2%*, against *6.2%* here. Changes in language, prompt, scoring or sampling may explain that difference; the observed loss is not a fixed model attribute.

## 8. Data and Reproduction

The artifacts accompanying the study are organized under `benchmarks/llm` in the project sources at `https://github.com/lupodevelop/emd`. They include the saved specifications, selected task identifiers, raw completed replies, scorer, protocol metadata and token-rate assumptions. The `v3` and `codex-v1` result directories distinguish the Claude collection from the incomplete OpenAI extension. The earlier run is retained separately.

Program scoring uses the emd compiler and VM, documented with their verification boundaries in the companion language paper [6]. The task inputs, normalization policy, step bound and missing-answer convention form part of this study's evaluation contract. The saved replies permit rescoring without new model calls.

## 9. Conclusion

The tested configurations can produce executable emd programs from a supplied specification: the complete Sol basic-prompt sample includes *70* program answers satisfying their task tests out of *72*. Exact reproduction of long dash strings is also often successful. A visually opaque source can therefore be a usable model output in this case, without implying that it is undecipherable by a person with the specification.

Functional success and accepted presentation remain different outcomes. The paired Claude data show substantial recoverable formatting loss for Haiku, reduced under an explicit decoder instruction; other configurations show smaller or no detectable gains. Reporting task content, immediately usable output and its token-cost estimate separately is the contribution of this case study, rather than a claim about AI's general ability or human readability.

## A. Supplementary Results

Table 4 breaks down Claude *bare* responses by task family. Tables 5 and 6 retain answer-level discordant counts alongside the task-level prompt comparisons. Every pair uses the same task and repetition; the answer-level McNemar values do not account for dependence between repetitions. The comparisons are exploratory and not adjusted for multiplicity.

Table 4: Claude condition *bare*, by task, pooling Haiku and Sonnet, in percent. $n$ is the number of answers.

| Kind | Tier | $n$ | Comp. | Strict | Cont. | Loss |
|---|---|---|---|---|---|---|
| complete | easy | 18 | 88.9 | 88.9 | 88.9 | 0.0 |
| complete | hard | 24 | 29.2 | 20.8 | 45.8 | 54.5 |
| copy | easy | 18 | – | 100.0 | 100.0 | 0.0 |
| copy | hard | 36 | – | 97.2 | 97.2 | 0.0 |
| distinguish | easy | 18 | – | 61.1 | 100.0 | 38.9 |
| distinguish | hard | 30 | – | 60.0 | 86.7 | 30.8 |
| explain | easy | 18 | – | 94.4 | 94.4 | 0.0 |
| explain | hard | 30 | – | 70.0 | 70.0 | 0.0 |
| generate | easy | 18 | 55.6 | 55.6 | 77.8 | 28.6 |
| generate | hard | 42 | 38.1 | 33.3 | 64.3 | 48.1 |
| repair | easy | 18 | 83.3 | 77.8 | 83.3 | 6.7 |
| repair | hard | 24 | 50.0 | 45.8 | 62.5 | 26.7 |

Table 5: Strict success against *bare*. Answers: pass only under that condition / only under *bare*. Tasks: better / worse.

| Model | Condition | Answers | $p$ | Tasks | $p$ |
|---|---|---|---|---|---|
| Haiku | explicit | 46 / 7 | <0.001 | 24 / 4 | <0.001 |
| Haiku | schema | 45 / 8 | <0.001 | 22 / 2 | <0.001 |
| Haiku | plain | 22 / 16 | 0.42 | 14 / 10 | 0.54 |
| Sonnet | explicit | 17 / 7 | 0.06 | 10 / 5 | 0.30 |
| Sonnet | schema | 19 / 11 | 0.20 | 13 / 7 | 0.26 |
| Sonnet | plain | 12 / 11 | 1.00 | 8 / 6 | 0.79 |
| All | explicit | 63 / 14 | <0.001 | 26 / 3 | <0.001 |
| All | schema | 64 / 19 | <0.001 | 24 / 5 | <0.001 |
| All | plain | 34 / 27 | 0.44 | 15 / 9 | 0.31 |

Table 6: Pairs of models under *bare*: answers where only the first / only the second succeeds.

| Pair | Strict | $p$ | Content | $p$ |
|---|---|---|---|---|
| Haiku vs Sonnet | 5 / 55 | <0.001 | 7 / 30 | <0.001 |

## References


[1] M. Chen et al., "Evaluating Large Language Models Trained on Code," arXiv:2107.03374v2, 2021.

[2] S. Lee and M. Kang, "LatentMD: Benchmarking Markdown Boundary Failures in LLM-Generated Text," arXiv:2609.06993v2, 2026.

[3] E. B. Wilson, "Probable Inference, the Law of Succession, and Statistical Inference," *J. American Statistical Association*, vol. 22, no. 158, pp. 209–212, 1927.

[4] Q. McNemar, "Note on the Sampling Error of the Difference Between Correlated Proportions or Percentages," *Psychometrika*, vol. 12, no. 2, pp. 153–157, 1947.

[5] OpenAI, "API Pricing," Standard text-token rates for GPT-6 Luna and GPT-6.1 Sol.

[6] D. Scaratti, "emd: A Language of Eight Unicode Dashes and How Far It Can Be Verified," preprint, Zenodo, 2026, doi:10.5281/zenodo.24011442.