

Maximizing First-Session Bit Rate with Short Nonword Targets

Allen Su

April 2026

Abstract

This report describes a 60-second browser task designed to maximize achieved bit rate under the Science Corporation SWE homework constraints. The task presents targets sampled independently and uniformly with replacement from a fixed alphabet, records exact-match selections, and reports bit rate as

$$B = \frac{\log_2(N - 1) \max(S_c - S_i, 0)}{t}.$$

I implemented and iteratively refined the task interface, input handling, Supabase logging pipeline, public analysis dashboard, and target alphabets, then ran a small pilot study over 11 candidate conditions. The deployed final task is available at <https://sciencetakehome.vercel.app>, and the public Supabase-backed dashboard is available at <https://sciencetakehome.vercel.app/admin>. The GitHub repository, including source code, cleaned pilot data, and analysis scripts, is available at <https://github.com/allensu02/sciencetakehome>. Across five subjects who completed all current conditions, real three-character words achieved the highest observed bit rate (mean 8.28 bps), but word targets are not appropriate for the final task constraint. Among nonword conditions, the strongest valid condition was a three-character nonword alphabet with $N = 16009$, which achieved 7.08 bps mean and 6.98 bps median. I therefore select three-character nonword targets as the final submission task: they preserve fast first-session typing while providing a large enough alphabet to carry substantially more information per correct selection than single letters.

1 Goal

The assignment is to build a game that maximizes achieved human bit rate in a single 60-second evaluation. The important design constraint is not just speed: targets must be sampled i.i.d. uniformly with replacement from a fixed alphabet of size $N \geq 3$. The sequence cannot contain exploitable patterns, predictive structure, language-model statistics, or word-level assistance.

This creates a direct engineering tradeoff. Larger N increases bits per correct selection through $\log_2(N - 1)$, but difficult targets reduce the number of correct selections S_c and increase incorrect selections S_i . The best task is therefore not necessarily the task with the largest alphabet. It is the task that maximizes the product of information per target and first-session human throughput.

2 Final Task

The final selected task uses fixed-length, three-character nonword targets. The alphabet contains all lowercase three-character strings after excluding real words and common word-list entries, giving

$$N = 16009.$$

Each trial displays one target string. The player types the target followed by space. A correct completed string increments S_c ; an incorrect key advances immediately to the next target and increments S_i . Targets are sampled uniformly with replacement from the fixed alphabet, and future targets do not depend on prior targets, user behavior, correctness, timing, or any language model.

This condition is a compromise between two useful extremes:

- Single letters are extremely fast but have only $N = 26$, so each correct selection carries only $\log_2(25) \approx 4.64$ bits.
- Very large five-character nonword alphabets carry many bits per selection, but the longer targets substantially reduce selections per minute.

For three-character nonwords with $N = 16009$, each correct selection carries $\log_2(16008) \approx 13.97$ bits, while the target remains short enough for first-session keyboard typing.

3 Implementation

The system is a TypeScript/React web app with a pure session engine. The engine owns target generation, input matching, score accounting, timing, and bit-rate computation. The UI displays a continuous stream of previous, current, and upcoming targets, with a running bit-rate readout updated during the session.

The final scored session lasts exactly 60 seconds. Timing starts on the first keystroke, not on page load, so the participant can visually orient before the scored window begins. At the end of the session the app reports N , S_c , S_i , and final bit rate.

For remote pilot collection, the app uploads completed JSON logs to Supabase. Each log contains the subject ID, condition ID, selected configuration, target presentations, task keystrokes, timestamps, and final metrics. A public dashboard at <https://sciencetakehome.vercel.app/admin> reads the logs, groups them by subject, and reconstructs bit-rate-over-time curves.

4 Interface Iteration

The final task was not the first version. The app evolved through several iterations driven by pilot observations and implementation constraints.

Continuous target stream. The earliest interface presented literally one target on the screen at a time. In pilots, this made the task feel like a sequence of isolated reactions: participants had to finish one target, wait for the next visual target to appear, reacquire it, and then begin typing again. I changed the interface to feel more like a typing test, inspired by Monkeytype, where the current target is highlighted in a continuous stream and upcoming targets are already visible. This lets participants keep their eyes in one stable region and prepare the next target while finishing the current one.

Remote logging and review. I added Supabase logging and a dashboard once the task became an empirical design search rather than a single static game. This made it possible to compare conditions across users, inspect sessions, and avoid relying on manual downloads.

54

jjf djj jgj fjj jjf hgk dds bjg dhf
 dgj kgh dgg dff jjf gjj adg bhh hgg
 dgj jgj jff fkk adf ddd djf jfh fhg

Figure 1: Monkeytype-inspired continuous target stream. The current target is highlighted while upcoming targets remain visible, reducing visual reacquisition between selections.

Target boundary handling. I briefly experimented with visual letter spacing, where letters were displayed with added visual separation but users did not type spaces. This made the display more complex without solving the main ambiguity. After asking users for feedback after sessions, Isabel and others mentioned that typing space after each word felt natural from ordinary typing-test habits, and that adding it would make the task clearer. I replaced the visual-only separation with an explicit required-space delimiter. The player now types each target followed by space, which makes target completion unambiguous for both the user and the scoring engine.

Error lockout. During informal pilots, an incorrect key sometimes caused a cascade: the app advanced to the next target immediately, and the user’s next keystroke could land on the new target before they had visually recovered. I added a 150 ms input lockout after an incorrect advancing error. This preserves the scoring rule while preventing one motor slip from creating multiple accidental errors.

Alphabet search. The initial alphabets were single characters, common three-character words, pronounceable nonwords, and common five-character words. I then added a QWERTY ergonomic scoring model that rewards home-row use, hand alternation, and inward rolls while penalizing weak fingers, same-finger transitions, and row jumps. This produced ergonomic word and nonword subsets. Finally, I expanded the condition set to include full three-character and large five-character nonword alphabets so the pilot could measure the information-throughput tradeoff directly.

5 Alphabet Construction

The ergonomic conditions were built from explicit candidate pools rather than from adaptive generation during the task. For three-character nonwords, I enumerated all 26^3 lowercase strings, removed entries found in the real-word and common-word exclusion lists, and ranked the remaining 16009 strings with a QWERTY cost model. The score adds per-key costs for row and finger difficulty, then adds transition costs for distance, row jumps, same-hand transitions, and same-finger reuse. It gives small bonuses for hand alternation and inward rolls. The $N = 100$ and $N = 1000$ three-character nonword conditions are the first prefixes of this ranking; the $N = 16009$ condition uses the full remaining set.

For five-character nonwords, the full set is too large to store directly in the session configuration. I therefore used two representations. The $N = 1000$ five-character condition is a stored ergonomic list generated from pronounceable CVCVC-style candidates. The $N = 100000$ and $N = 11871011$ conditions use a generated alphabet: strings are drawn by rank from an ergonomic character order and rejected if they appear in the exclusion lists. In all cases, once an alphabet size is chosen, targets are sampled uniformly over that fixed alphabet. The ranking determines which strings are eligible for a condition; it does not make easier strings more likely within that condition.

6 Pilot Evaluation

The Supabase database contains 111 session rows across 15 subject IDs. Because the condition set evolved during development, the cleanest analysis uses the five subjects who completed all 11 current conditions. Each condition was a 60-second scored session. For subjects with repeated runs of a condition, I used the latest run for that subject-condition pair. The cleaned session-level dataset and analysis script are included in the repository under `data/` and `analysis/`.

I recruited friends as pilot subjects and conducted most sessions over Zoom. I joined a call with each subject, asked them to share their screen while they ran the browser task, and recorded the session so I could review interface confusion, error cascades, and other qualitative behavior after the fact. These recordings were used only as design feedback rather than as part of the numerical bit-rate analysis. Recordings can be provided on request.

The 11 current conditions span characters, real words, and nonwords:

- single characters with $N = 26$;
- three-character common words with $N = 100$;
- all three-character real words with $N = 1567$;
- three-character nonwords at $N = 100$, $N = 1000$, and $N = 16009$;
- five-character common words with $N = 1000$;
- all five-character real words with $N = 10365$;
- five-character nonwords at $N = 1000$, $N = 100000$, and $N = 11871011$.

7 Results

Table 1 summarizes the complete-subject analysis. The best raw condition used real three-character words; the best nonword condition used the full three-character nonword alphabet.

The highest observed bit rate came from all three-character real words. This condition ranked first for four of five complete subjects and ranked in the top three for all five. This is useful as an empirical ceiling: lexical familiarity helps substantially.

However, the final task should avoid word targets. Among nonword conditions, $N = 16009$ three-character nonwords performed best. It achieved 7.08 bps mean, 6.98 bps median, and an average within-subject rank of 3.0. It also had the best nonword rank profile: it was first overall for one subject and top-three overall for three of five subjects.

The five-character conditions show why maximizing N alone is not enough. The largest five-character nonword alphabet has $N = 11871011$, which gives $\log_2(N - 1) \approx 23.5$ bits per correct target. But participants completed only 19.2 correct selections on average, so its mean bit rate was

Condition	N	Mean bps	Median bps	Mean S_c	Mean S_i	Accuracy
Single characters	26	6.16	6.66	84.4	4.8	95.8%
3-char common words	100	6.56	6.41	68.4	9.0	88.2%
3-char all words	1567	8.28	9.02	53.4	6.6	89.8%
3-char nonwords	100	3.43	3.43	36.4	5.4	88.5%
3-char nonwords	1000	4.75	4.48	35.8	7.2	86.5%
3-char nonwords	16009	7.08	6.98	37.0	6.6	86.3%
5-char common words	1000	5.25	5.31	40.8	9.2	81.7%
5-char all words	10365	5.07	6.23	33.4	10.6	77.1%
5-char nonwords	1000	3.22	3.65	25.8	6.4	82.9%
5-char nonwords	100000	4.26	4.43	20.0	4.6	83.2%
5-char nonwords	11871011	5.80	6.27	19.2	4.4	83.6%

Table 1: Pilot results for the five subjects who completed all current conditions. Accuracy is $S_c/(S_c + S_i)$.

5.80 bps. In contrast, the three-character nonword condition has fewer bits per target but nearly twice as many correct selections.

Why three characters beat one. At first glance, the best three-character nonword result is surprising. Three single-letter selections carry $3\log_2(25) \approx 13.93$ bits, almost the same as one three-character nonword selection, which carries $\log_2(16008) \approx 13.97$ bits. If a three-character target took exactly three times as long as a one-character target, the two designs should have similar bit rates.

The pilot data suggest that typing time did not scale linearly with target length. Single-character trials were very fast, but each target still required a separate visual acquisition, target boundary, space delimiter, and transition to the next item. Three-character targets amortized those fixed per-target costs over roughly three times as many bits. In the complete-subject data, participants made 84.4 correct single-character selections on average versus 37.0 correct selections for full three-character nonwords. That is only a $2.28\times$ reduction in correct selection rate, while the information per correct selection increased by about $3.01\times$. After incorrect selections are subtracted, the effective rate was still favorable: $S_c - S_i$ was 79.6 for single characters and 30.4 for three-character nonwords, a $2.62\times$ reduction. The extra entropy per target outweighed that slowdown.

My hypothesis is that three-character strings are short enough to be typed as one small motor chunk, especially in the continuous stream interface, while single letters pay too much overhead per scored selection. Five-character strings move past that sweet spot: they provide more bits per target, but become slow enough that the added entropy no longer compensates.

8 Why Not Use Real Words?

The all-words condition is the strongest raw performer, but it is not the right final task. The homework explicitly rules out word-level targets and exploitable language structure. Real words are easier because humans bring prior language familiarity, chunking, and motor memory. That advantage is exactly what makes the result scientifically interesting, but it also makes the condition less appropriate as the final submission.

I therefore treat real-word conditions as controls:

- common words measure performance with a small familiar alphabet;
- all three-character words estimate the upper bound from lexical familiarity;
- nonwords measure performance when lexical familiarity is removed.

Under that interpretation, the final task should use nonwords even though real words scored higher in the pilot.

9 Final Selection

I select the three-character nonword condition with $N = 16009$ for the final task. It is the best valid balance of target length and alphabet size:

- It avoids real word targets.
- It uses a fixed alphabet sampled uniformly with replacement.
- It has no adaptive sequencing or predictive text.
- It preserves short, quickly learnable keyboard actions.
- It outperforms all other nonword conditions in the complete-subject pilot.

The central result is that first-session bit rate is limited more by human target execution than by theoretical alphabet entropy. Three-character nonwords sit near the useful point on that curve: large enough to make each correct target valuable, but short enough that users can still complete many targets in 60 seconds.

10 Limitations

The pilot sample is small, with only five complete subjects for the current 11-condition comparison. The condition set also evolved over time, so earlier subjects did not see all final conditions. Orders were not fully counterbalanced, though the complete subjects did not all run the conditions in the same order. The participants were not the Science Corporation grading panel, so the final score may differ, especially for a very fast typist.

The nonword alphabet depends on dictionary and frequency-list exclusion. Some obscure real words may remain, and some excluded strings may not be familiar to most users. For the purposes of the task, the important property is that the target alphabet is fixed, nonadaptive, and sampled uniformly; nevertheless, the exact word-exclusion procedure should be documented.

Finally, the current pilot data are quantitative logs only. Session recordings could add useful qualitative evidence about visual confusion, recovery after errors, and fatigue, but I did not use them in the numerical analysis above.

11 Conclusion

The best raw pilot condition was all three-character words, but the best compliant final task is three-character nonwords with $N = 16009$. This condition maintains the assignment’s i.i.d. fixed-alphabet requirement while achieving the strongest nonword bit rate observed in the pilot. The result supports a simple design principle: for first-session human bit-rate tasks, short targets with a moderately large nonword alphabet outperform both low-entropy single-character tasks and very large but slow five-character alphabets.