

AI Coding CLIs on a Subscription Budget

Free sample. Chapters 1, 5, and 9 of *The 2026 Bench Report*. The full edition adds the methodology, the full results matrix, the thinking-tier economics chapter, the multi-turn failure autopsy, and three appendices (batch ledger, grader's rulebook, data provenance).

1. Executive summary, or: three paragraphs if that's all you read

I ran 85 gradable coding tasks (plus 10 multi-turn patch jobs and a blind-knowledge track) through the same model family at three settings: glm-5.3 on thinking:low, glm-5.3 on thinking:high, and glm-4-flash. The cheap setting won. thinking:low scored 83/85 (97.6%), thinking:high scored 80/85 (94.1%), and flash scored 47/85 (55.3%). The expensive tier was also 24-52% slower and burned 24-46% more tokens. Across six batches I could not find a single problem where thinking:high genuinely beat low. Not one.

The interesting result is where flash dies. It isn't on the hard problems. Lock-free ordering, deque tricks, decorator rewriting — the classic "hard" interview material with a known recipe — flash passes those at 80-100%. What kills it is medium-difficulty work with constraints that have no template: regexes with negative lookarounds, CSV quoting, config parsing where the spec says "unknown key must raise ValueError." Twenty medium IO/regex problems. Flash passed four. The failure mode isn't stupidity; it's that flash skims the constraint list the way a tired senior skims a ticket at 6pm.

And the failure that should scare you doesn't show up in any single-run score. In the multi-turn patch test, flash was asked to modify a working `slugify` function. It rewrote the function, met the new requirement, and silently dropped the hyphen from the old behavior — `slugify('Hello World')` now returned `'helloworld'`. The code looked clean. Only running the tests caught it. That is the most expensive class of bug in day-to-day AI-assisted work: the model you trusted to patch your code un-patches something you already shipped.

If you take nothing else from this book: run the cheap tier by default, force single-code-block output when you must run the expensive tier, always run the code before you trust it, and never hand unreviewed "cheap batch work" to a small model without a verification step.

5. Where flash dies (a map)

38 flash failures. The big model at low tier passed all 38 of those same problems. Zero cases of the reverse — the one record that looked like it (q71) was a grader misfire, caught by the tokenize rule and re-judged. So this is a one-way gap, which makes the autopsy clean.

Three death patterns:

A. Crashes on delivery. q79 references an undefined variable inside a closure, two different uses, `NameError` both ways. Batch 5's q58 runs `findall` with a capture group and then treats the resulting tuples as strings. Six batch 4 problems forget an import or call `.split` on a list. The first line of the program is where it dies.

B. Constraint lists, partially executed. q72 and q74 use `.get` with a default where the spec demanded `ValueError` on unknown keys. q66 and q67 leave a literal `0` in the function body where a named constant belonged; q69 converts the threshold but not the return value. Batch 5's q54 sorts by insertion order when the spec said lexicographic. The spec was a checklist; flash read it like a vibe.

C. Statically fine, dynamically wrong. Batch 5's q59 stacks negative lookarounds into a regex that contradicts itself and matches no legal IP. q82 writes CSV with unquoted comma fields. q65 calls BSD `sed -i` on macOS, which errors, while also violating the spec's no-in-place-edit rule.

By category: regex 8 of 13 dead (batch 5 regex went 0/5), file handling 6 of 13, constant-refactoring 3 of 13. Meanwhile the template-shaped hard problems — lock ordering, dequeues, decorators — flash passes at 80-100%.

The gap isn't difficulty. It's whether the problem has a canonical recipe to copy. "Hard with a known shape" beats "medium with no shape" every time for a small model.

9. The decision tree

You have a coding task. Start here.

- |
- |— Is it a one-file script, function, or small tool?
 - |— YES → cheap tier (thinking:low). Medium-and-below was 100% here, fastest and cheapest. Done.
- |
- |— Hard refactor, multi-constraint concurrency, gnarly logic?
 - |— Still low. Hard tier: 94%. The expensive tier costs more, runs slower, and scored worse.
- |
- |— Some rule forces you onto the high tier (org policy, whatever)?
 - |— Add to the prompt: "Output a single complete code block only. No additional example blocks or alternative versions."
And set `max_tokens ≥ 8192`.
- |
- |— Big batch of cheap work – redaction, format conversion, log parsing?
 - |— Don't hand it to flash unsupervised: it passed 20% of the IO/regex batch, and one rework eats the entire saving.
 - |— If you do use flash: template-shaped work only (known algorithms, known concurrency patterns), and run every output. No exceptions.
- |
- |— Patching existing code with a requirement change?
 - |— low tier. flash's one multi-turn failure was losing existing behavior while adding the new one.
- |
- |— Any AI-written code, any model, any tier?
 - |— Run the core path for real – open it, click it, test it.
Static review is a filter, not a verdict. Proven twice in this data.