

NYC AI CO-WORKING · 2026-10-29

Asking Agents To Port Codebases

Kevin Scott



OVERVIEW

Asking Agents to Port Codebases

01 Overview

02 Methodology

03 Results



Asking Agents to Port Codebases

Small & Reproducible

- Porting at scale gets the attention
- I want reproducible ports: one reference library that spawns the others
- SDKs. Open source libraries.

Experiment

- 3 experiments
- 617 ports of the same library, both directions
- Compare against hand-written implementations

Codebase

- [gbnf](#): a string library for GBNF grammars
- Two implementations, written by hand
 - Python, 1,305 lines
 - TypeScript, 1,129 lines
- Language-agnostic tests



Methodology



METHODOLOGY · GBNF

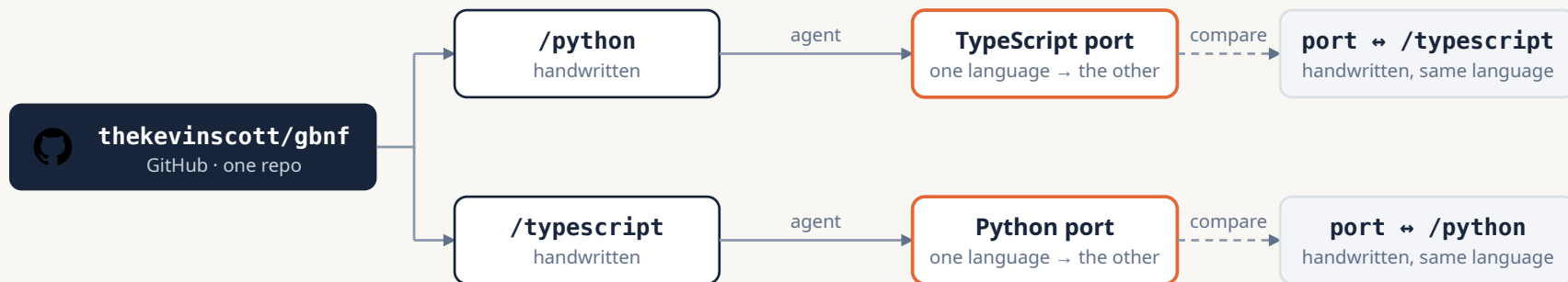
Fairly language-agnostic

Mostly parity between the languages

Written by hand

Language-agnostic integration tests

Python: 1,305 lines of code in 42 files · TypeScript: 1,129 lines in 42 files · tests excluded



What the agent gets

THE CONTAINER

- Locked-down Docker container
- No network other than LLM endpoints. No npm, no PyPI: gbnf is published on both.
- pytest, vitest, uv, pnpm pre-installed

ON DISK

```
/input/      the reference, read-only
python/gbnf/          35 files
python/tests/generated/ 5 files
/target/      empty
```

PROMPT †

You are porting software from one language to another.

The reference is mounted at `/input`.

Write the port to `/target`.

The library itself may specify additional porting instructions below. Paths in them are relative to `/input`.

Port the implementation under `python/` to `typescript`.
There is no network access. uv, pytest, pnpm and vitest are installed.

The following test suites are included:

- python integration tests: `python/tests/generated/`

* yellow = varies per run: the source directory, the target language, and which test suites the agent is told it has.

† Verbatim from run 20261012T021310Z_595388f6 (python → typescript, source integration tests on), the prompt the 192 tests-configuration runs received. Later runs got a longer one: the reference is marked read-only and the two Dockerfiles the sandbox image was built from are pasted in.



Which model, effort and tests give a port that works, and at what cost?

Three experiments, one library, both directions. Each port: one agent, one sandbox, the reference read-only, no network.

V1 · pilot

Opus 5, high effort. Vary which tests the agent sees. Found the `__pycache__` leak.

4 conditions × 2 directions · **42 ports**
· \$286

V2 · models × effort

Fable, Opus, Sonnet, Haiku × low, medium, high, very high. Unit and source tests always on.

16 cells × 2 directions × 12 runs · **383 ports** · \$1,528

V2 · tests

Opus 5.5, medium effort. Every combination of unit, source and target tests on or off.

8 conditions × 2 directions × 12 runs
· **192 ports** · \$355



"Cheating": hard to prevent, harder to define

“Let me extract the original test expectations from the leftover `.pyc` test files — they're a real spec for behavior.”

What happened

V1 withheld the tests. A host-side pytest run had left them compiled in `__pycache__`, mounted read-only beside the source. 8 runs had them. 4 decompiled them. 3 shipped the recovered tests.

Was that cheating?

The agent broke no rule. It read a file in `/input`, where we said the reference lives. The boundary was ours to draw, and we drew it wrong.

`__pycache__`

bytecode holds full source

`.pytest_cache`

lists the failing test ids

npm · PyPI

gbnf is published on both

sourcemaps

json2gbnf bundles gbnf's source

training data

the repo is public



Results



RESULTS · OVERALL STATISTICS

Wall time

Tokens

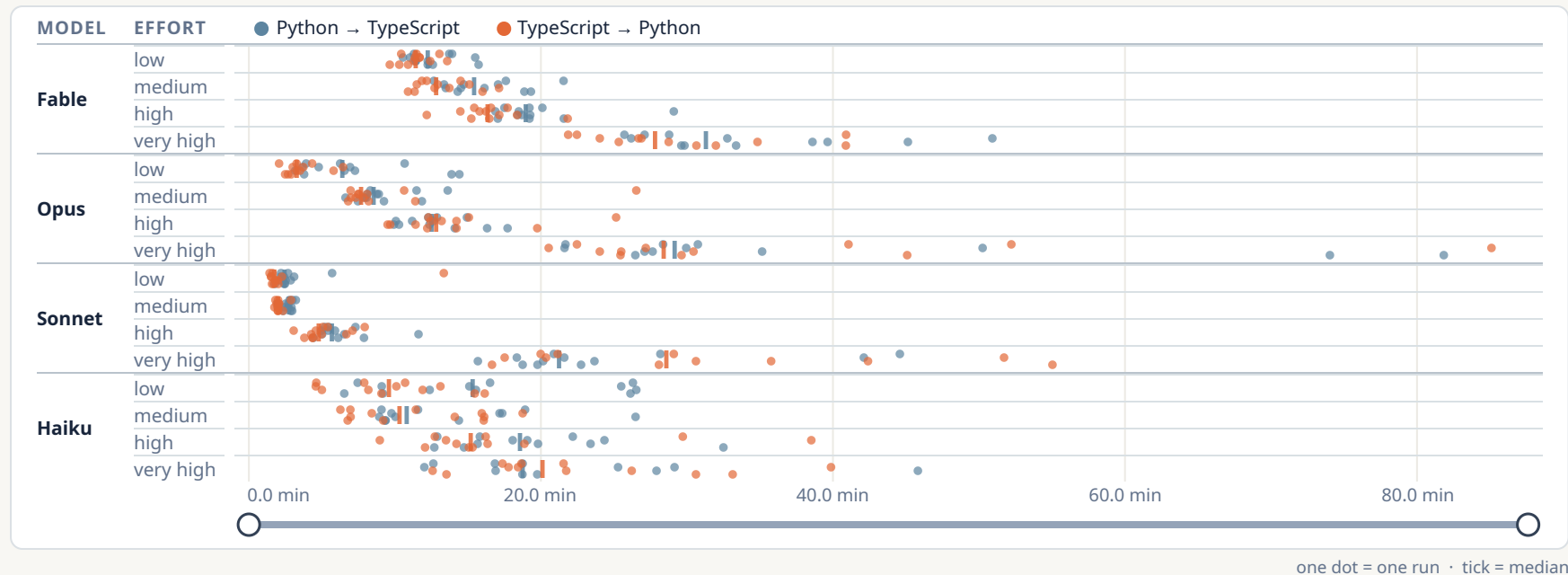
Cost

Files

LOC

MODELS × EFFORT

TESTS EXPERIMENT



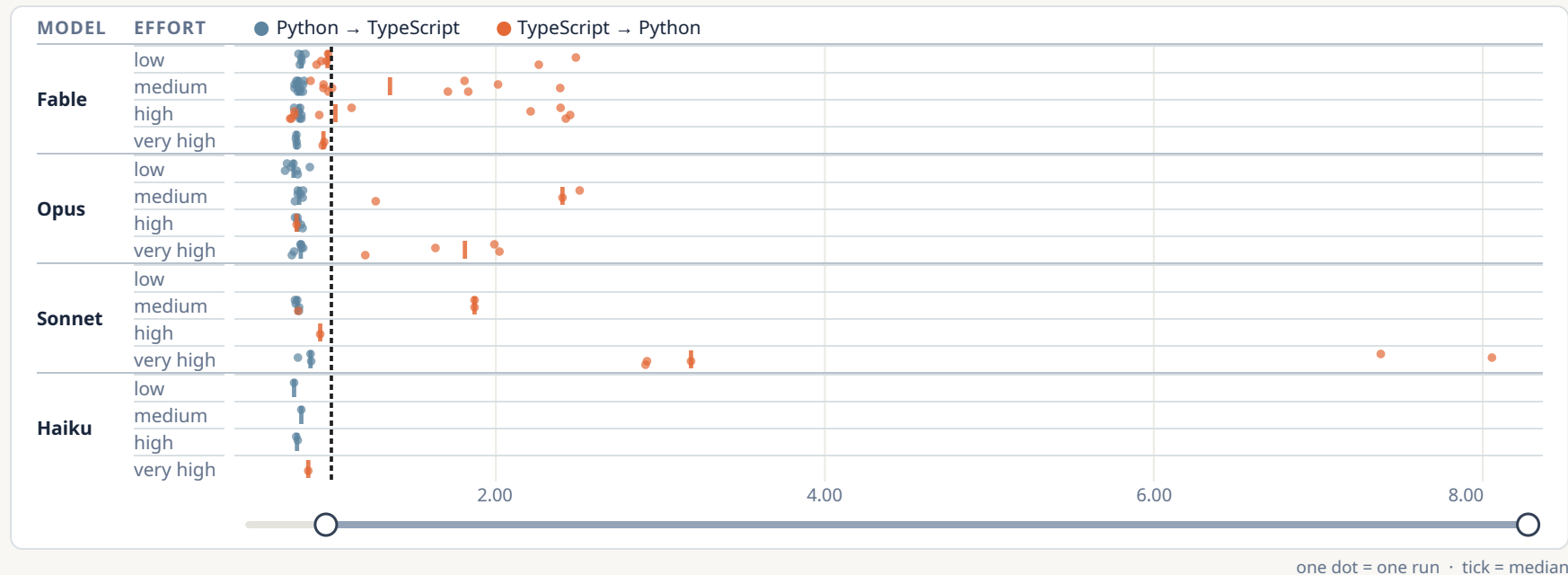
models × effort: 4 models × 4 efforts · tests: 8 test conditions, Opus medium · n=12 per cell and direction · cost = transcript API calls at list rates,
scripts/cost_metrics.py · 2026-10-29



RESULTS · SPEED OF THE PORTED LIBRARY

MODELS × EFFORT

TESTS EXPERIMENT



tests experiment: 137 of 192 ports timed (89 py→ts, 48 ts→py; the rest would not load) · models × effort: 111 of 383 ports timed so far (65 py→ts, 46 ts→py; 174 would not load, the rest still running) · ladder = 22 cases, [analysis/exercise-api/cases/ladder.jsonl](#) · [scripts/ladder_metrics.py](#) · 2026-10-29

RESULTS · SIMILARITY TO THE HANDWRITTEN REFERENCE

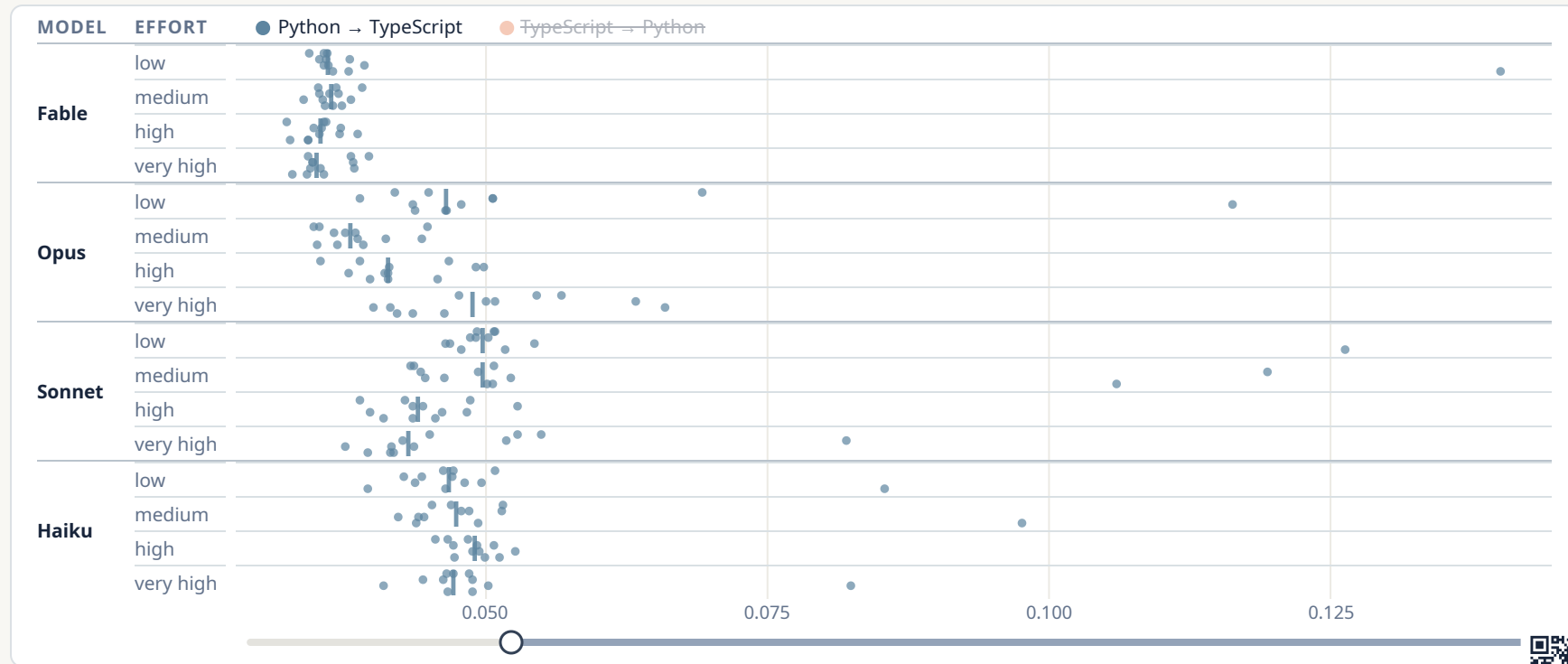
Embedding distance

Lines shared with reference %

Diff lines vs reference

MODELS × EFFORT

TESTS EXPERIMENT



one dot = one run · tick = 1



models × effort: 4 models × 4 efforts · tests: 8 test conditions, Opus medium · n=12 per cell and direction · measure-code-distance git-diff · measure-embedding · 2026-10-29

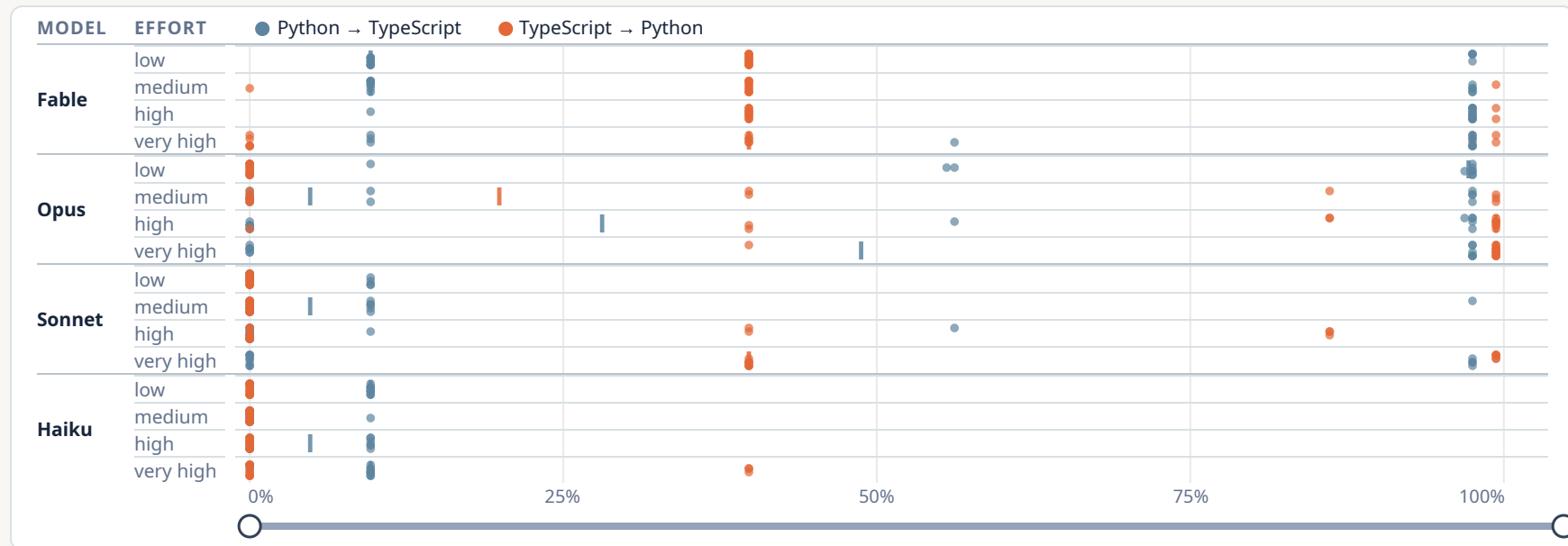
RESULTS · HELD-OUT TESTS · HOW OFTEN THE PORT AGREES WITH THE REFERENCE

Behaviour %

API %

MODELS × EFFORT

TESTS EXPERIMENT



one dot = one run · tick = median



Takeaways



TAKEAWAYS

Three things the 617 ports say

1 For the money: Opus.

\$5 – 9 per port that runs, at medium or high effort. Haiku never produced a port that ran. Fable ran more often
Python → TypeScript and cost 1.2 – 11× more per working port.

2 The agent has a house style. Handwritten implementations are the outlier.

Two ports of the same setting share 61 – 66 % of their lines. A port and the handwritten version share 33 – 38 %.

3 Target-language integration tests are cheap insurance. Source-language tests on their own are not enough, and can hurt.

Target tests did not beat no tests: 88 – 95 % held-out API agreement either way. But they were the only tests that told the agent the interface. Source-language tests alone: 7 – 8 of 12 ports would not load. Target tests undid that, for fewer tokens than no tests.



TAKEAWAY 1 · WHICH MODEL AND EFFORT, FOR THE MONEY?

Opus: the cheapest port that runs

model	effort	Python → TypeScript		TypeScript → Python	
		\$ per port that ran	ran	\$ per port that ran	ran
Fable	low	\$20	4 of 12	–	0 of 12
Fable	medium	\$20	5 of 12	\$89	1 of 12
Fable	high	\$11	11 of 12	\$52	2 of 12
Fable	very high	\$22	8 of 12	\$83	2 of 12
Opus	low	\$3	7 of 12	–	0 of 12
Opus	medium	\$9	4 of 12	\$8	4 of 12
Opus	high	\$9	5 of 12	\$5	9 of 12
Opus	very high	\$17	6 of 12	\$9	11 of 12
Sonnet	low	–	0 of 12	–	0 of 12
Sonnet	medium	\$7	1 of 12	–	0 of 12
Sonnet	high	–	0 of 12	\$4	3 of 12
Sonnet	very high	\$11	4 of 12	\$9	5 of 12
Haiku	low	–	0 of 12	–	0 of 12
Haiku	medium	–	0 of 12	–	0 of 12
Haiku	high	–	0 of 12	–	0 of 12
Haiku	very high	–	0 of 11	–	0 of 12

\$ per port that ran = the cell's total spend ÷ ports the held-out driver could load and run, so failed attempts are priced in. Orange = under \$1



TAKEAWAY 2 · THE AGENT HAS A HOUSE STYLE. HANDWRITTEN IMPLEMENTATIONS ARE THE OUTLIER.

Ports agree with each other more than with the handwritten port

distance between	Python → TypeScript	TypeScript → Python
two ports, same model and effort	0.015	0.028
two ports, same model, different effort	0.017	0.034
two ports, different models	0.022	0.050
a port and the handwritten implementation	0.044	0.068

Median embedding distance, lower = closer. Every pair of the 383 ports in a direction (18,145 / 18,336 pairs), and each port against the handwritten implementation (191 / 192 ports). Closest cross-model pair of models: Fable ↔ Opus at 0.018 (Python → TypeScript), Haiku ↔ Sonnet at 0.040 (TypeScript → Python).



TAKEAWAY 3 · TARGET-LANGUAGE INTEGRATION TESTS ARE CHEAP INSURANCE. SOURCE-LANGUAGE TESTS ON THEIR OWN ARE NOT ENOUGH, AND CAN HURT.

Held-out API agreement by the test suites the agent was given

tests the agent was given	Python → TypeScript		TypeScript → Python	
	API agreement	loaded	API agreement	loaded
no tests	88 %	12 of 12	95 %	12 of 12
source integration	0 %	5 of 12	0 %	4 of 12
target integration	88 %	11 of 12	95 %	11 of 12
source + target integration	87 %	7 of 12	95 %	12 of 12

Median held-out API agreement over 12 runs per cell, Opus medium: the share of the held-out driver's calls the port answers like the handwritten implementation. A port that did not load scores 0 %. Orange = most ports in the cell would not load. Target tests do not beat no tests. They are the only condition that told the agent the interface the driver expects; given source tests alone, the agent chose its own package layout and the driver could not import it. Target integration tests also cost fewer tokens than no tests.



Slides: **thekevinscott.com/talks/2026/october/nyc-ai-coworking/slides**

Data and code: github.com/thekevinscott/ports-research



"Cheating": hard to prevent, harder to define

“Let me extract the original test expectations from the leftover `.pyc` test files — they're a real spec for behavior.”

What happened

V1 withheld the tests. A host-side pytest run had left them compiled in `__pycache__`, mounted read-only beside the source. 8 runs had them. 4 decompiled them. 3 shipped the recovered tests.

Was that cheating?

The agent broke no rule. It read a file in `/input`, where we said the reference lives. The boundary was ours to draw, and we drew it wrong.

`__pycache__`

bytecode holds full source

`.pytest_cache`

lists the failing test ids

npm · PyPI

gbnf is published on both

sourcemaps

json2gbnf bundles gbnf's source

training data

the repo is public



APPENDIX · DISTANCE BETWEEN PORTS · DO REPEATS OF ONE SETTING AGREE WITH EACH OTHER?

MODELS × EFFORT

TESTS EXPERIMENT

Python → TypeScript

	low	medium	high	very high
Fable	751	734	598	618
Opus	1254	905	972	1145
Sonnet	1078	2151	712	1529
Haiku	878	1018	1016	944

TypeScript → Python

	low	medium	high	very high
Fable	826	912	1024	1154
Opus	1732	1112	1279	1465
Sonnet	914	510	777	1262
Haiku	1384	1250	1234	1308



APPENDIX · HELD-OUT TESTS · WHEN A PORT SCORES 0, WHAT FAILED?

model	effort	Python → TypeScript			TypeScript → Python		
		not loaded	loaded, failing	driven	not loaded	loaded, failing	driven
Fable	low	0	8	4 / 12	0	12	0 / 12
Fable	medium	0	7	5 / 12	1	10	1 / 12
Fable	high	0	1	11 / 12	0	10	2 / 12
Fable	very high	0	4	8 / 12	5	5	2 / 12
Opus	low	2	3	7 / 12	12	0	0 / 12
Opus	medium	6	2	4 / 12	6	2	4 / 12
Opus	high	6	1	5 / 12	1	2	9 / 12
Opus	very high	6	0	6 / 12	0	1	11 / 12
Sonnet	low	8	4	0 / 12	12	0	0 / 12
Sonnet	medium	6	5	1 / 12	12	0	0 / 12
Sonnet	high	10	2	0 / 12	7	2	3 / 12
Sonnet	very high	8	0	4 / 12	0	7	5 / 12
Haiku	low	3	9	0 / 12	12	0	0 / 12
Haiku	medium	11	1	0 / 12	12	0	0 / 12
Haiku	high	6	6	0 / 12	12	0	0 / 12
Haiku	very high	3	8	0 / 11	9	3	0 / 12

Over 413 runs: 102 driven · 123 loaded but failing · 188 not loaded. Driven by effort, both directions pooled (n = 96 each, 95 for very high): low 11 · medium 15 · high 30 · very high 36. The behaviour and API slide plots every run. A 0 is a port the driver could not run. A loaded port that accepted nothing the reference accepts lands on a floor: 39.8 % TypeScript → Python (60 ports), 9.6 % Python → TypeScript (58 ports). The 3 median on that slide is this floor, not partial correctness.

v2-models-effort · max runs on their own slide · `public/models-effort-status.csv` from `scripts/status_metrics.py` · not loaded = every held-out pair crashed · loaded, failing = the port accepted none of the inputs the reference accepts · driven = otherwise · 2026-10-29



Three mechanisms, all at the interface

status	direction	what the driver saw	ports	by model
not loaded	TypeScript → Python	cannot import name 'RuleEnd'	54	Sonnet 31, Opus 18, Haiku 5
not loaded	TypeScript → Python	No module named 'gbnf'	48	Haiku 41, Fable 6, Opus 1
not loaded	Python → TypeScript	no src/index.ts	86	Sonnet 33, Haiku 30, Opus 23
loaded, failing	TypeScript → Python	every input raises	58	Fable 41, Sonnet 9, Opus 6, Haiku 2
loaded, failing	TypeScript → Python	mixed	2	Haiku 2
loaded, failing	Python → TypeScript	every input raises	58	Haiku 26, Fable 19, Sonnet 10, Opus 3
loaded, failing	Python → TypeScript	rejects every input	5	Opus 3, Sonnet 1, Fable 1

driven ports	n	at the reference-bug ceiling	below it
Fable	38	38	–
Opus	49	44	3 at 86.1%, 2 at 96.9%
Sonnet	15	12	3 at 86.1%
Haiku	0	0	–

The driver imports `gbnf`, calls `GBNF(grammar, input)` and iterates for `RuleEnd`; in TypeScript it imports the default export of `src/index.ts`. A package nested under `python/` or `src/`, a renamed `RuleEnd`, or a class export that needs `new` scores 0 with no grammar parsed. Fable's Python ports define `RuleEnd` as a `TypedDict`, so the driver's `isinstance` raises on every accepted input. Ceiling 97.5 % (779 / 799) and 99.4 % (794 / 799); the rest are the target reference's own bugs.



Layout follows the prompt

direction	top of the port	ports	by model
models × effort · long prompt · 413 ports			
TypeScript → Python	gbnf/ at the top (the reference layout)	154	Sonnet 50, Opus 48, Fable 47, Haiku 9
TypeScript → Python	python/...	35	Haiku 33, Fable 2
TypeScript → Python	src/...	13	Haiku 8, Fable 4, Opus 1
Python → TypeScript	src/index.ts present	125	Fable 52, Opus 31, Haiku 26, Sonnet 16
Python → TypeScript	no src/index.ts	86	Sonnet 33, Haiku 30, Opus 23
tests experiment · short prompt · Opus medium · 192 ports			
TypeScript → Python	gbnf/ at the top (the reference layout)	83	of 96
TypeScript → Python	src/... or other	13	of 96
Python → TypeScript	src/index.ts present	89	of 96
Python → TypeScript	no src/index.ts	7	of 96

The reference lives at `/input/python/gbnf` and `/input/javascript/src`. Similarity and the held-out driver match paths against that layout, so a port that keeps the `python/` prefix, or puts the TypeScript under `gbnf/` instead of `src/`, scores near 0 before a line is compared. Haiku's 1–3 % TypeScript → Python similarity is this. The one cell both experiments share, Opus medium with unit + source tests: 6 of 12 Python → TypeScript ports lack `src/index.ts` under the long prompt, 0 of 12 under the short one. Part of "not loaded" on the held-out slide is the prompt, not the model.



Move the folder, fix the case, re-measure

direction	model	misplaced ports	as scored	folder moved	also renamed	final
TypeScript → Python	Fable	6	6.3	33.7	0	33.7
TypeScript → Python	Opus	1	0.3	32.1	0	32.1
TypeScript → Python	Haiku	41	1.3	34.5	0	34.5
Python → TypeScript	Opus	23	8.0	8.3	23	37.5
Python → TypeScript	Sonnet	33	1.2	1.3	33	27.0
Python → TypeScript	Haiku	30	8.7	36.3	6	37.4

The 134 ports whose package is not where the reference keeps it, copied to scratch and re-scored with the same git-diff measure after two mechanical fixes, nothing inside any file changed. First the package directory is moved (`python/gbnf` → `gbnf` , `gbnf` → `src` , and so on). That alone lifts every TypeScript → Python port to 32–35 %. 62 Python → TypeScript ports (all Sonnet, all Opus, 6 Haiku) still match nothing because they name folders `rules_builder/` where the reference has `rules-builder/` ; lower-casing and dashing those names lifts them to 27–38 %. Medians in the "final" column use the renamed score where a rename was applied. The lowest of the 134 after both fixes is 20.0 %.



max against very high: 30 runs so far, 1 to 9 per cell

direction	model	effort	n	wall min	tokens	cost \$	LOC	diff lines	behaviour %	API %	driven
Python → TypeScript	Fable	very high	12	31.3	5,205,744	14.36	1,036	1,676	97.5	89.7	8 / 12
Python → TypeScript	Fable	max	4	51.2	4,349,127	17.35	1,022	1,698	97.5	89.7	4 / 4
Python → TypeScript	Opus	very high	12	29.2	11,234,867	8.63	1,113	2,282	48.7	43.9	6 / 12
Python → TypeScript	Opus	max	6	80.7	34,634,241	24.57	1,058	2,470	48.7	43.9	3 / 6
Python → TypeScript	Sonnet	very high	12	21.2	10,325,860	3.65	1,062	2,857	0.0	0.0	4 / 12
Python → TypeScript	Sonnet	max	1	60.2	53,142,672	11.42	1,323	3,375	0.0	0.0	0 / 1
Python → TypeScript	Haiku	very high	11	18.8	15,810,158	1.49	1,013	1,816	9.6	0.0	0 / 11
Python → TypeScript	Haiku	max	9	28.4	30,016,627	2.71	1,042	2,642	0.0	0.0	0 / 9
TypeScript → Python	Fable	very high	12	27.8	5,196,706	13.10	1,070	2,191	39.8	9.0	2 / 12
TypeScript → Python	Fable	max	5	45.3	5,951,717	18.49	1,116	2,312	39.8	9.0	1 / 5
TypeScript → Python	Opus	very high	12	28.4	13,311,610	8.15	1,205	2,268	99.4	94.9	11 / 12
TypeScript → Python	Opus	max	1	83.9	28,038,511	14.88	1,192	2,358	39.8	9.9	0 / 1
TypeScript → Python	Sonnet	very high	12	28.6	12,013,940	3.71	1,144	2,198	39.8	9.0	5 / 12
TypeScript → Python	Sonnet	max	2	116.1	55,064,600	12.00	1,390	2,610	99.4	94.9	2 / 2
TypeScript → Python	Haiku	very high	12	20.1	16,866,758	1.64	970	2,940	0.0	0.0	0 / 12
TypeScript → Python	Haiku	max	2	50.7	46,758,099	5.82	1,045	2,637	19.9	4.5	0 / 2

Medians over the runs that exist. max roughly doubles wall time and spend over very high for every model; it does not move size, diff lines or the held-out scores on this evidence. The charts hide max because its cells are 1 to 9 runs against 12 everywhere else.



The long prompt

```
You are porting software from one language to
another.
The reference is mounted read-only at `/input`.
Write the port to `/target`.
There is no network access.
The sandbox image was built from these two
Dockerfiles, the second on top of the first:
[Dockerfile 1, 23 lines: node:24 base, Claude Code
2.1.296, socat, botbox-relay]
[Dockerfile 2, 20 lines: uv, Python 3.14.3, pytest,
vitest, pnpm]
The library itself may specify additional porting
instructions below. Paths in them are relative to
`/input`.
Port the implementation under python/ to
typescript.
The following test suites are included:
- python unit tests: python/gbnf/ (colocated,
*_test.py)
- python integration tests: python/tests/generated/
```

AGAINST THE SHORT ONE (SLIDE 5)

- 2,170 characters against about 450. The two Dockerfiles are 1,565 of them.
- "mounted read-only" is new. The short prompt said only "mounted at /input".
- "There is no network access" moved up; the tool list (uv, pytest, pnpm, vitest) left the prose for the Dockerfiles.
- Two suites listed (unit, source integration) instead of one.
- The Dockerfiles changed twice during the batch (botbox 0.0.20 → 0.1.1 → 0.1.72) and a memory cap was added; all three recorded in the experiment's DEVIATIONS.md.



APPENDIX · WHAT THE STUDY TOOK

	runs	spend	tokens	agent time
models × effort (incl. 30 max)	413	\$1,912	3,007 M	130 h
Fable	105	\$1,081	426 M	39 h
Opus	103	\$536	777 M	37 h
Sonnet	99	\$173	527 M	20 h
Haiku	106	\$122	1,277 M	34 h
tests experiment (Opus medium)	192	\$355	411 M	23 h
both	605	\$2,267	3,418 M	153 h
models × effort spend by what the held-out driver could do with the port				
driver ran the port	102	\$804	782 M	46 h
loaded, failing	123	\$676	815 M	39 h
not loaded	188	\$432	1,410 M	46 h

Spend is every API call in the transcripts priced at Claude API list rates, summed; tokens are input, output and cache reads and writes from the transcripts. Agent time is wall-clock inside the container, summed, not calendar time: batches ran in parallel over about two weeks. Haiku used the most tokens of any model and the least money.

driven ports	Python → TypeScript	TypeScript → Python
Fable	32 / 52	6 / 53
Opus	25 / 54	24 / 49
Sonnet	5 / 49	10 / 50
Haiku	0 / 56	0 / 50

