CSC 347 - Concepts of Programming Languages

Token Efficiency

Instructor: Stefan Mitsch



Learning Objectives

- Compare token usage across different languages

Coding Agent Token Usage

Wu, Z., Anderson, C., Guha, A.: The Best Programming Language for Tokenmaxxing

Anderson, M.: Which Programming Languages are Most Token Efficient

- Does the choice of programming language affect AI coding costs?
- AI coding agents work iteratively (each iteration consumes tokens)

1. Understand a programming problem
2. Plan a solution
3. Generate code
4. Compile and execute it
5. Inspect failures
6. Revise and repeat

- Final program may be short, but substantial reasoning and debugging may be necessary to produce it

Experimental Setup

4 languages

- Python: Widely used
- Java: Mainstream OO language
- Rust: Systems language
- OCaml: Functional language

5 models

- Qwen3.6-35B-A3B
- Gemma-4-26B-A4B
- GLM-5.2
- GPT-5.5, Claude Sonnet 4.6

Problems

- 100-problem subset of LiveCodeBench
- Agent framework: mini-swe-agent
- Up to 40 agent turns per problem
- 10-minute limit per session
- Agents inspect test results and iteratively improve solutions

Main Results

- Language changes token cost
- Median output token consumption, multipliers relative to Python

Model	Java	Rust	OCaml
Gemma	0.85×	1.07×	1.44×
Qwen	1.21×	1.20×	1.69×
GLM	1.34×	1.57×	1.54×
GPT	1.3×	1.36×	1.3×
Claude	1.18×	1.16×	1.28×

Where are the Tokens Spent?

- Agents continue even when they have a working solution that passes all tests
  - **Before** solution: implementation, debugging, until problem solved (all tests pass)
  - **After** solution: optimization, refactoring, verification, unnecessary revisions
- Results (examples)

Gemma

Language	Tokens before	Tokens after
Python	1,743	3,554
Java	2,454	4,546
Rust	2,026	7,311
OCaml	1,760	1,173

GLM

Language	Tokens before	Tokens after
Python	785	117
Java	729	97
Rust	1,574	122
OCaml	1,374	109

Sonnet

Language	Tokens before	Tokens after
Python	878	103
Java	1,149	95
Rust	1,134	136
OCaml	1,484	136

- ❗ Proprietary models tend to submit solution more directly



Why do Agents Struggle with some Languages?

- Problem is often syntax, not algorithms
  - Breakthrough: All tests pass
  - Improved: Some previously failing tests pass
  - Stay: Same tests pass before and after
  - Regressed: Some passing tests now fail
  - Churn: Some tests improve while others regress
  - Broke: Previously passing tests all fail
  - Stuck: Test outcomes remain unchanged
- Languages with little (public) examples fail more often in compilation (stuck)
- When compilation fails, next iteration has no additional test feedback



Patterns Behind Token Waste

- **Planning in comments**: agents sometimes develop algorithms inside comments before code (comments become temporary workspace, perhaps to avoid information dropping from context)
- **Solve in Python, then translate**: agents sometimes first use a popular language, then attempt to translate (less idiomatic solution in target language)
- **Second-guessing**: agents often question solutions that are functionally correct (are cases missing, efficiency improvement, more elegant implementation?)
- **Testing and self-examination**: agents write additional test cases

❗ Token usage is partly a language problem and partly an agent-control problem



Research Methodology

Instead of evaluating the final program, the paper:

- Captures every intermediate solution
- Re-executes each snapshot against the benchmark tests
- Records the pass/fail status of individual tests
- Compares consecutive snapshots
- Agent actions
  - Investigate approaches
  - Write code
  - Rewrite/replace an approach
- Agent actions
  - Execute plan
  - Fix known bug
  - Debug/investigate unknown error
  - Fix build/tool setup
  - Revert to earlier code
  - Verify correctness
  - Handle additional inputs
  - Improve speed or memory
  - Cosmetic refactor



LLM-Based Program Synthesis

- Coding agents and LLMs can help solve programming language problems
- Can Large Language Models Autoformalize Kinematics?
  - Fig. 1: agent pipeline
  - Fig. 2: results with vs. without iteration
- LLM-Powered Automatic Theorem Proving and Synthesis for Hybrid Games
  - Fig. 2: theorem proving agents
  - Fig. 4: synthesis agents
  - Appendix F: prompts
- 🔧 Read papers, what processes are the agents following?
- 🔧 Read the prompts in the appendix, what elements and themes stick out?



Summary

Practical implications when developing agentic solvers:

- Provide language-specific reasoning and debugging hints to agents
- Introduce stopping conditions after verified success
- Measure token costs alongside correctness
- Study intermediate results, not just final code

> The best programming language for an AI coding agent is not necessarily the one that produces the shortest code. It is the one in which the agent can reliably reach a correct solution with the least unnecessary reasoning and revision.