BACHELOR THESIS · CSE3000 RESEARCH PROJECT

Can Small Beat Big?

Evaluating Fine-Tuned CodeT5 Models on Assertion Generation Quality and Eficiency

TUDelft

Thimo van Leeuwen

Supervisors: Annibale Panichella, Mitchell Olsthoorn

EEMCS, Delft University of Technology

1 The Problem

- Test Assertions confirm the code does what it should. Writing good ones is demanding.
- Fine-tuned code models can write them. **AsserT5** (Fine-Tuned CodeT5-large, 770M) leads by exact-match accuracy [1].
- Developer want efficiency&fault-detection capability

The assertion-generation task. Example from Primbs [1].

FOCAL METHOD

```
char last(String s) {
  return s[s.length-1];
}
```

TEST METHOD · ASSERTION MASKED

```
@Test void testLast() {
  char res = last("abc");
<ASSERTION>
}
```

CodeT5 generates ▾

GENERATED ORACLE

```
assertEquals(res, 'c');
```

2 Research Questions

RQ1 — SIZE WITHIN THE FAMILY

How does model size in the fine-tuned CodeT5 family affect assertion generation quality and efficiency?

RQ2 — SPECIALIZATION vs SCALE

How do fine-tuned CodeT5 models compare to larger code-specific LLMs on assertion generation quality and efficiency (**Qwen2.5-Coder** 3B / 7B / 14B)?

3 Methodology

01 Dataset

- **541 assertion tasks** from **10 real-world Java projects** (GitBug-Java), filtered from 55.
- Each task masks one assertion; kept with its *test prefix* and a heuristic **focal method**.

02 Prompt construction

- CodeT5: `focal <SEP> masked test` in a tight **386-token** window (106 inputs truncated).
- Qwen: same context, 32K window — no truncation needed.

03 Inference

- CodeT5: Pytorch, top-1 prediction only.
- Qwen: 4-bit via Ollama, nondeterministic — **averaged over 6 runs**.

04 Mutation analysis

- **PIT** (ALL operators) vs. a *human* and a *no-oracle* baseline.
- Generated assertion appended; scored only if it compiles & executes.

SIX MODELS EVALUATED	PARAMS	MEMORY
FINE-TUNED CodeT5 (FP32)		
small	60M	**0.24 GB**
base	220M	0.87 GB
large	770M	2.89 GB
Qwen2.5-Coder (4-bit)		
3B	3B	1.90 GB
7B	7B	4.70 GB
14B	14B	9.00 GB

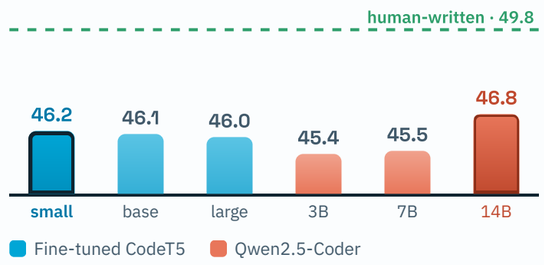
Staged quality funnel — each metric a precondition for the next

Compilation rate → assertion compiles when appended · **Execution validity** → it passes on the correct program · **Mutation score** → share of injected faults killed *(primary)*. Efficiency is measured by **inference time** and **memory footprint**.

4 Results

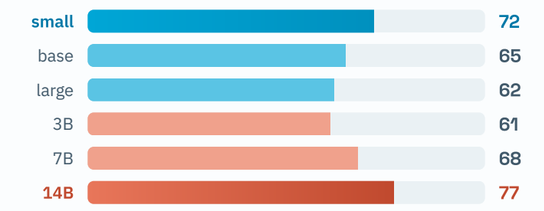
Average mutation score

% of mutants killed · 10 projects · higher is better. Bars start at 44, not 0, which is the mutation score of test prefix only.



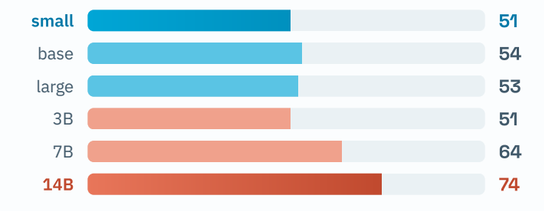
Average compilation rate

% of generated assertions that compile · higher is better



Average execution validity rate

% of compiled assertions that pass on the correct program



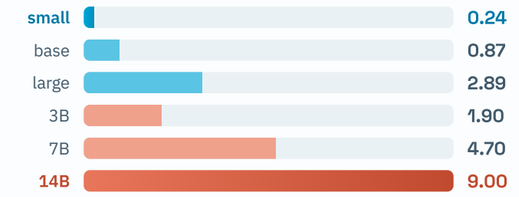
Reading the results

- Within CodeT5, mutation scores are **near-identical** (46.0–46.2) — the **smallest is marginally best**.
- All CodeT5 variants **beat Qwen-3B & 7B**
- Only **Qwen-14B** leads by **0.6 pp**, and that edge sits in just **2 of 10** projects.

5 Discussion

Memory footprint of weights

GB held in memory during inference · lower is better



Median inference time / assertion

0.21s	0.83s	0.54s
CodeT5-small	CodeT5-large	Qwen-14B

4× faster than large · **2.6× faster than 14B**

CodeT5-small compiled more often due to staying closer to the test prefix and not assuming non-existent symbols exist in somecases where CodeT5-base and CodeT5-large did.

CodeT5-small vs. CodeT5-large: it gives the same quality at **4× faster** inference and **12× less memory**; vs. Qwen-14B it trades just **0.6 pp** of mutation score for **2.6× faster** inference and **38× less memory** (0.24 vs. 9.00 GB).

Moreover, when looking at the union of mutant killed between Qwen-14B and CodeT5-small, CodeT5-small recovers 73% of the mutants killed.

Qwen-14B advantage is in two projects; one memorized constant recall, and better hashcode computation reconstruction from the focal method using the assertion.

6 Conclusion

- CodeT5 achieves most of the performance of Qwen-14B, at 38x smaller memory footprint and a median inference time of 0.21s.

For practitioners on local hardware, we recommend fine-tuned CodeT5-small.

[1] S. Primbs et al., "AsserT5: Test Assertion Generation Using a Fine-Tuned Code Language Model," *AST*, 2025.