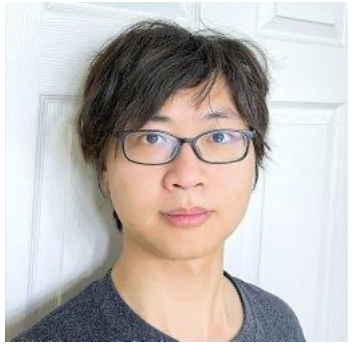
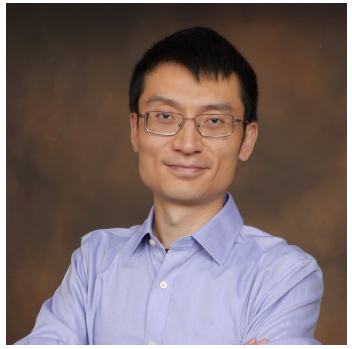
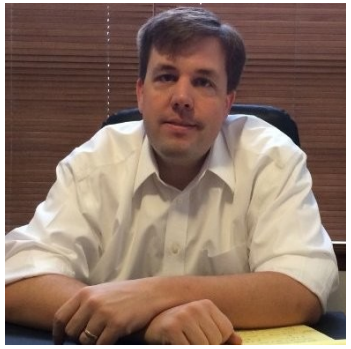
How Efficient is LLM Generated Code?
A Rigorous & High-Standard Benchmark



Ruizhong Qiu
UIUC
(Presenter)



Weiliang Will Zeng
Qualcomm



James Ezick
Qualcomm

Christopher Lott
Qualcomm

Hanghang Tong
UIUC

https://github.com/q-rz/enamel

ACKNOWLEDGEMENTS

How Efficient is LLM-Generated Code?

HumanEval: $2^{\Theta(n)}$ recursions

```
def fib(n):
  if n == 0:
    return 0
  if n == 1:
    return 1
  return fib(n - 1) + fib(n - 2)
```

GPT-4 Turbo: $\Theta(n)$ iterations

```
def fib(n):
  a, b = 0, 1
  for _ in range(n):
    a, b = b, a + b
  return a
```

Ours: $\Theta(\log n)$ iterations

```
def fib(n):
  if n == 0: return 0
  a, b = 0, 1
  for n in bin(n)[3 :]:
    a, b = a * a + b * b, b * (a * 2 + b)
    if n == '1': a, b = b, a + b
  return b
```

- Example problem (from [1]): computing the n-th Fibonacci number.
  - While all **correct**, the three implementations above have **different efficiencies**.
  - **Efficiency** is **crucial** in real-world applications but is largely **overlooked** in existing benchmarks.
  - How does **LLM-generated** code compare with **expert-written** code in terms of **efficiency**?

[1] Chen et al. Evaluating large language models trained on code. *arXiv*: 2107.03374, 2021.

Proposed Level-Based Evaluation

HumanEval: $2^{\Theta(n)}$ recursions

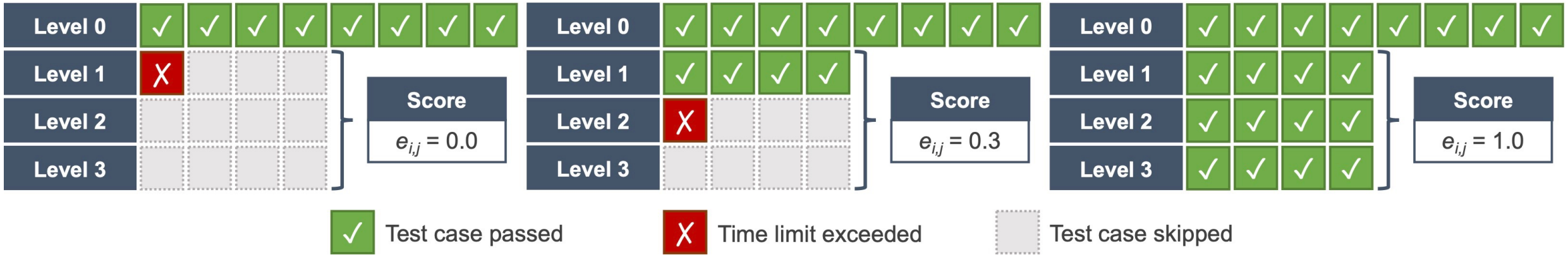
```
def fib(n):
  if n == 0:
    return 0
  if n == 1:
    return 1
  return fib(n - 1) + fib(n - 2)
```

GPT-4 Turbo: $\Theta(n)$ iterations

```
def fib(n):
  a, b = 0, 1
  for _ in range(n):
    a, b = b, a + b
  return a
```

Ours: $\Theta(\log n)$ iterations

```
def fib(n):
  if n == 0: return 0
  a, b = 0, 1
  for n in bin(n)[3 :]:
    a, b = a * a + b * b, b * (a * 2 + b)
    if n == '1': a, b = b, a + b
  return b
```



Efficiency Score of a Code Sample

- Notations:
  - T_i: time limit of problem i. $c_{i,j}$: the j-th LLM-generated code sample of problem i.
  - $t_{i,j,l,m}$: the execution time of code $c_{i,j}$ for the m-th test case in level l.
  - $t^*_{i,l,m}$: the execution time of our reference solution of problem i for the m-th test case in level l.
  - L: number of levels. M_l: number of test cases of level l. h_l: hardness of level l.
- Efficiency score of code $c_{i,j}$ in level l:

$$f_{i,j,l} := \frac{(T_i - \max\{t_{i,j,l,m}\}_{m=1}^{M_l})^+}{T_i - \max\{t^*_{i,l,m}\}_{m=1}^{M_l}}$$

- Efficiency score of code $c_{i,j}$:

$$e_{i,j} := \begin{cases} \frac{\sum_{l=1}^{L} h_l \cdot f_{i,j,l}}{\sum_{l=1}^{L} h_l}, & \text{if code } c_{i,j} \text{ is correct;} \\ 0, & \text{otherwise.} \end{cases}$$

Efficiency Metric for an LLM

- Proposed efficiency metric: a **generalization** of the widely used pass@k metric [1].

$$\mathrm{eff}_i@k := \mathop{\mathbb{E}}_{c_{i,1},\ldots,c_{i,k}\sim \mathrm{LLM}(z_i)} \left[\max_{j=1}^{k} e_{i,j} \right]$$

- **Unbiased** & **variance-reduced** estimator (Theorem 1):

$$\widehat{\mathrm{eff}}_i@k := \mathop{\mathbb{E}}_{\substack{J\subseteq\{1,\ldots,n\} \\ |J|=k}} \left[\max_{j\in J} e_{i,j} \right] = \sum_{r=k}^{n} \frac{\binom{r-1}{k-1}}{\binom{n}{k}} e_{i,(r)}$$

- **Numerically stable** implementation:

Algorithm 1 Numerically stable $\widehat{\mathrm{eff}}_i@k$

Input: score list $[e_{i,1}, \ldots, e_{i,n}]$; the target k

Output: the estimated $\widehat{\mathrm{eff}}_i@k$

$\lambda_n \leftarrow \frac{k}{n}$
for $r \leftarrow n-1, n-2, \ldots, k$ **do**
  $\lambda_r \leftarrow \lambda_{r+1} \cdot \left(1 - \frac{k-1}{r}\right)$
end for
$[e_{i,(1)}, \ldots, e_{i,(n)}] \leftarrow \mathrm{sort}([e_{i,1}, \ldots, e_{i,n}])$
return $\sum_{r=k}^{n} \lambda_r e_{i,(r)}$

[1] Chen et al. Evaluating large language models trained on code. *arXiv*: 2107.03374, 2021.

Benchmark Development

ID	Problem Description	HumanEval+ Solution	Our Expert Solution
#10	Find the shortest palindrome that begins with a given string S	$O(\|S\|^2)$: Enumerate suffixes and check palindromicity	$\Theta(\|S\|)$: Use Knuth–Morris–Pratt w.r.t. reversed S plus S
#36	Count digit 7's in positive integers $< n$ that are divisible by 11 or 13	$\Theta(n \log n)$: Enumerate integers $< n$ and count the digits	$\Theta(\log n)$: Design a dynamic programming over digits
#40	Check if a list l has three distinct elements that sum to 0	$O(\|l\|^3)$: Enumerate triples in l and check their sums	$O(\|l\|^2)$: Use a hash set and enumerate pairs in l
#109	Check if a list a can be made non-decreasing using only rotations	$O(\|a\|^2)$: Enumerate the rotations of a and check	$O(\|a\|)$: Check if the list a has at most one inversion
#154	Check if any rotation of a string b is a substring of a string a	$O(\|b\|^2\|a\|)$: Enumerate rotations and run string matching	$O(\|a\| + \|b\|)$: Run the suffix automaton of a w.r.t. $b + b$

- Problemset: **142** selected problems from HumanEval [1].
  - Including two subsets: (i) ***Algorithm Design***; (ii) ***Implementation Optimization***.

- **Efficient** reference solutions & **strong** test case generators:
  - We employ a **human expert** to curate them.
  - **Significantly** outperformed the canonical solutions in **HumanEval** [1] / **HumanEval+** [2].

Name	eff@1	pass@1
HumanEval	0.455	0.908
HumanEval+	0.513	0.972
ENAMEL (ours)	1.000	1.000

[1] Chen et al. Evaluating large language models trained on code. *arXiv*: 2107.03374, 2021.
[2] Liu et al. Is your code generated by ChatGPT really correct? Rigorous evaluation of large language models for code generation. *NeurIPS Datasets & Benchmarks Track*, 2023.

Main Results (Truncated)

- Even strong LLMs still **fall short** of generating **expert-level** efficient code.

Model	Greedy		Sampling					
	eff@1	pass@1	eff@1	pass@1	eff@10	pass@10	eff@100	pass@100
GPT-4 Turbo	0.470	0.796	—	—	—	—	—	—
GPT-4	0.454	0.831	—	—	—	—	—	—
Llama 3 70B Instruct	0.421	0.746	0.438	0.747	0.526	0.836	0.575	0.880
Llama 3 8B Instruct	0.344	0.592	0.345	0.564	0.500	0.770	0.595	0.874
Mixtral 8x22B Instruct	0.408	0.746	0.407	0.721	0.575	0.870	0.704	0.923
Mixtral 8x7B Instruct	0.266	0.444	0.279	0.456	0.436	0.689	0.542	0.810
Claude 3 Opus	0.401	0.789	—	—	—	—	—	—
Claude 3 Sonnet	0.345	0.662	0.365	0.677	0.498	0.814	0.594	0.887
Claude 3 Haiku	0.386	0.739	0.382	0.730	0.478	0.831	0.529	0.861
Phind Code Llama V2	0.394	0.683	0.372	0.638	0.584	0.862	0.723	0.935
ChatGPT	0.364	0.683	0.374	0.673	0.557	0.847	0.690	0.937
Code Llama 70B Python	0.264	0.500	0.082	0.177	0.326	0.610	0.614	0.908
Code Llama 34B Python	0.268	0.458	0.226	0.405	0.511	0.786	0.711	0.934
Code Llama 13B Python	0.216	0.408	0.204	0.372	0.487	0.732	0.714	0.899
Code Llama 7B Python	0.247	0.373	0.180	0.320	0.432	0.663	0.643	0.837
StarCoder	0.195	0.352	0.134	0.236	0.355	0.557	0.542	0.787
CodeGen 16B	0.169	0.310	0.122	0.219	0.326	0.512	0.536	0.761
CodeGen 6B	0.193	0.296	0.111	0.188	0.298	0.455	0.491	0.694
CodeGen 2B	0.153	0.254	0.098	0.168	0.264	0.389	0.421	0.602
CodeT5+ 16B	0.160	0.317	0.130	0.250	0.343	0.551	0.551	0.785
Mistral 7B	0.152	0.275	0.116	0.222	0.335	0.541	0.557	0.791

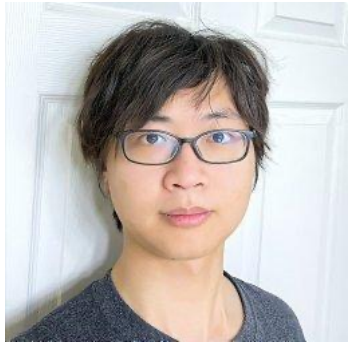
Evaluation on Two Subsets (Truncated)

- LLMs **struggle** in designing advanced algorithms.
- LLMs are largely **unaware** of implementation optimization.

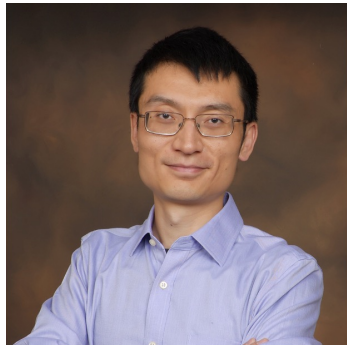
Model	Algorithm Design Subset						Implementation Optimization Subset					
	eff@1	pass@1	eff@10	pass@10	eff@100	pass@100	eff@1	pass@1	eff@10	pass@10	eff@100	pass@100
Llama 3 70B Instruct	0.246	0.660	0.306	0.749	0.359	0.750	0.404	0.791	0.497	0.869	0.551	0.920
Llama 3 8B Instruct	0.201	0.518	0.303	0.724	0.367	0.849	0.313	0.582	0.468	0.806	0.571	0.906
Mixtral 8x22B Instruct	0.225	0.635	0.363	0.837	0.470	0.900	0.376	0.783	0.556	0.914	0.686	0.947
Mixtral 8x7B Instruct	0.124	0.391	0.244	0.681	0.344	0.850	0.248	0.473	0.411	0.699	0.515	0.827
Claude 3 Sonnet	0.184	0.577	0.328	0.804	0.450	0.950	0.358	0.723	0.475	0.846	0.548	0.893
Claude 3 Haiku	0.149	0.692	0.208	0.752	0.266	0.775	0.360	0.772	0.465	0.889	0.513	0.923
Phind Code Llama V2	0.185	0.554	0.353	0.789	0.401	0.849	0.351	0.712	0.567	0.901	0.732	0.968
ChatGPT	0.120	0.488	0.304	0.799	0.483	0.950	0.337	0.715	0.508	0.864	0.633	0.949
Code Llama 70B Python	0.018	0.100	0.129	0.519	0.402	0.950	0.076	0.181	0.294	0.627	0.589	0.920
Code Llama 34B Python	0.071	0.293	0.271	0.713	0.425	0.881	0.197	0.415	0.473	0.804	0.687	0.949
Code Llama 13B Python	0.058	0.212	0.276	0.665	0.478	0.844	0.176	0.405	0.476	0.784	0.715	0.928
Code Llama 7B Python	0.068	0.202	0.231	0.589	0.393	0.761	0.165	0.349	0.417	0.703	0.620	0.863

Thanks for watching

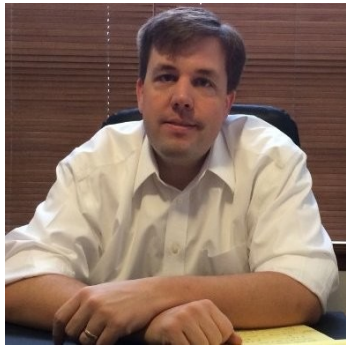
How Efficient is LLM Generated Code? A Rigorous & High-Standard Benchmark



Ruizhong Qiu
UIUC
(Presenter)



Weiliang Will Zeng
Qualcomm



James Ezick
Qualcomm

Christopher Lott
Qualcomm

Hanghang Tong
UIUC

https://github.com/q-rz/enamel

ACKNOWLEDGEMENTS