



08.30.2023

# ON THE EFFICACY OF LARGE LANGUAGE MODELS AS MODELS OF COMPUTATION

SE-691-682

Sebastian Just, Frank Ottey, Greg Morgan,  
Trevor Pawlewicz

# Introduction

LANGUAGE MODELS AS MODELS OF COMPUTATION

# Language Models as Programmers

- As autoregressive LLMs based on the transformer architecture have increased in capability, expectations around their efficacy at modeling real-world phenomenon
- The transformer architecture itself has proven a valuable architecture for domain-based sequence-to-sequence modeling – See OpenAI's Whisper model for an example in the ASR domain
- Our original thought: Could we apply language modeling techniques (autoregressive LLMs, transformer architecture) to computational modeling?
  - Original title: “Evaluating the efficacy of multi-task attention networks in detecting vulnerabilities in WASM byte code”
- Specifically, can LLM's be used to intuit understanding around what Web Assembly byte code does and means?

The screenshot shows a chat window with a dark theme. At the top, a user prompt says: "Write C++ code to check the given number is prime number or not." Below it, the AI response says: "Here's one way to implement it in C++:". The response includes a C++ code block with a function `isPrime` and a `main` function that takes user input and prints whether it's a prime number. A "Copy code" button is visible next to the code block. At the bottom of the chat window, there is a "Regenerate response" button and a text input field.

```
c
#include <iostream>

using namespace std;

bool isPrime(int n) {
    if (n <= 1) return false;
    for (int i = 2; i < n; i++) {
        if (n % i == 0) return false;
    }
    return true;
}

int main() {
    int n;
    cout << "Enter a number: ";
    if (isPrime(n)) {
        cout << n << " is a prime number." << endl;
    } else {
        cout << n << " is not a prime number." << endl;
    }
    return 0;
}
```

# What is Web Assembly?

- Web Assembly bytecode is a byte code format similar to JVM bytecode that can be run by a variety of VM implementations, both on a server or in a sandboxed browser
- As a bytecode format it is platform agnostic
- It is a popular target for performance-sensitive web-application code that might need to run on the client-side.
- Because WASM bytecode is run on the client-side – (in a user's browser) there is an inherent risk / required amount of trust from the user to run the code.
- Web Assembly is growing to become an isomorphic platform target: the Web Assembly System Interface (WASI) specification defines a runtime for server-side WebAssembly

```
(module
  (type $t0 (func))
  (type $t1 (func (param i32)))
  (type $t2 (func (param i32) (result i64)))
  (type $t3 (func (param i32 i32)))
  (type $t4 (func (param i32 i32) (result i32)))
  (type $t5 (func (param i32 i32) (result i64)))
  (type $t6 (func (param i32) (result i32)))
  (type $t7 (func (param i32 i32 i32) (result i32)))
  (type $t8 (func (param i32 i32 i32 i32) (result i32)))
  (type $t9 (func (result i32)))
  (type $t10 (func (param i32 i32 i32)))
  (type $t11 (func (param i32 i32 i32) (result i64)))
  (type $t12 (func (param i32 i32 i32 i32)))
  (type $t13 (func (param i32 i32 i32 i32 i32) (result i32)))
  (type $t14 (func (param i32 i32 i32 i32 i32 i32)))
  (type $t15 (func (param i32 i32 i32 i32 i32)))
  (type $t16 (func (param i32 i32 i32 i32 i32 i32) (result i32)))
  (type $t17 (func (param i64 i32 i32) (result i32)))
  (import "wasi_snapshot_preview1" "fd_write" (func $_ZN4wasi13lib_generated22wasi_snap
  (import "wasi_snapshot_preview1" "environ_sizes_get" (func $__wasi_environ_sizes_get
  (import "wasi_snapshot_preview1" "proc_exit" (func $__wasi_proc_exit (type $t1)))
  (import "wasi_snapshot_preview1" "environ_get" (func $__wasi_environ_get (type $t4)))
```

# Why use LLMs for Bytecode Analysis?

- LLM's have shown initial promise at generating and evaluating code.
- Analysis of code is generally performed either statically or dynamically
- Static analysis tends to have less reliable coverage while dynamic analysis tends to be non-performant. (Compiler Analysis vs. Valgrind, Sanitizers)
- Dynamic analysis requires running code, which, if the code is untrusted / malicious is already problematic
- Use LLMs to analyze bytecode fragments before execution by the VM?

## Similar / Prior Work

- "Deep Neural Networks for Malicious JavaScript Detection Using Bytecode Sequences," 2020 International Joint Conference on Neural Networks (IJCNN), Glasgow, UK, 2020, pp. 1-8, doi: 10.1109/IJCNN48605.2020.9207134.)
- <https://ieeexplore.ieee.org/document/9207134>
- Compiled V8 bytecode vs. WASM bytecode
- Linked functions included in VM compilation
- V8 bytecode then encoded via word2vec, featurized and used in a classifier

## A Contemporary Adaptation

- Instead of operating on V8 bytecode, which is specific to Chrome's V8 Javascript engine, start with WASM bytecode, which is platform-agnostic
- Instead of using word2vec, utilize modern auto-regressive LLMs (ChatGPT, Llama etc.) to perform bytecode encoding.
- Follow similar experiments with featurization and classification
- Present evaluation / results
- Use-case: As more performance-sensitive code may be required to run client-side for privacy and security reasons, there is a need to have tools to automatically verify the transmitted code as non-malicious

# Challenges

LANGUAGE MODELS VS. MODELS OF COMPUTATION



# Key Finding: Tokenization is Everything

- Tokenization greatly changes model output
- Tokenization greatly changes model performance
- Tokenization is contextual
- Tokenization is only as good as the input training data
- Obscure input sequences will be harder to tokenize and harder to get quality output from.

```

left = [x for x in arr if x < pivot]
middle = [x for x in arr if x == pivot]
right = [x for x in arr if x > pivot]

return quick_sort(left) + middle + quick_sort(right)

```

Clear

Show example

Tokens

103

Characters

267

```

def quick_sort(arr):
    if len(arr) <= 1:
        return arr

    pivot = arr[len(arr) // 2]

    left = [x for x in arr if x < pivot]
    middle = [x for x in arr if x == pivot]
    right = [x for x in arr if x > pivot]

    return quick_sort(left) + middle + quick_sort(right)

```

TEXT

TOKEN IDS

# Computation is not Autoregressive

- LLM's are trained in an auto-regressive style
- Computation models are not auto-regressive
- Token mis-predictions / inaccurate predictions are more likely to produce poor quality results (Computation is stricter / less expressive than language)

# The Limits of Transfer Learning and Dataset Availability

- Tokenizer had little experience with WASM sequences.
- Transfer learning options limited due to dataset availability
- Ability to load / run models impacted by computational availability

# Evaluating the Divergence of Language and Computation?

- How effective are LLM's at being "programmers" in the small case?
- How effective are LLM's at translating a given computation between languages?
- To what extent are LLMs simply understanding the semantic meaning of the tokens vs. understanding computation?
- How effective are LLM's at understanding computation given the changing context of language?

# Evaluating LLMs with WebAssembly

Assessing LLM's current ability to generate and interpret WebAssembly Text Code

Can LLM's generate WebAssembly Text (WAT) given sample C++ code?

Can LLM's interpret WebAssembly Text (WAT) and generate equivalent C++ code?

### Methodology:

1. We evaluated ChatGPT 3.5 using basic C++ programs and their compiled WAT representation:

- Sum of Two Integers
- Fibonacci
- Factorial
- Is Integer a Prime Number

2. Ask ChatGPT 3.5 to perform code translation using the following prompts:

- Can you translate this C++ to WebAssembly text? <C++ code>
- Can you translate this WebAssembly text to C++? <WAT code>

3. Compile and Run the resulting code

- Using [WasmFiddle](#) to compile and run C++
- Using [WasmExplorer](#) to compile and run WAT

### Evaluation Criteria:

- **Readable:** Does the code generated look reasonable to the human eye?
- **Compilable:** Does the code compile?
- **Runnable:** Does the code run?
- **Usable:** Does the generated code actually work as expected and give us the right answers?

# Sample – Factorial (C++ to WAT)

## Chat GPT Output

```
(module
  (func (param i32) (result i32)
    (local i32)

    ;; int factorial(int n)
    (local.set 1 (i32.const 1))      ;; result = 1
    (loop
      (local.get 0)                  ;; n
      (i32.const 1)
      (i32.gt_s)
      (br_if 1)                      ;; n > 1
                                      ;; break loop if n <= 1

      (local.get 1)                  ;; result
      (local.get 0)                  ;; n
      (i32.mul)
      (local.tee 1)
      (i32.const 1)
      (i32.sub)
      (local.set 0)                  ;; result
                                      ;; n
                                      ;; result = result * n

      ;; n -= 1
      (i32.sub)
      (local.set 0)
      (br 0)                          ;; n = n - 1
                                      ;; continue loop
    )

    ;; return result
    (local.get 1)
  )

  (func (result i32)
    ;; int main()
    (i32.const 5)
    (call 0)
  )
)
```

## Actual WebAssembly

```
(module
  (table 0 anyfunc)
  (memory $0 1)
  (export "memory" (memory $0))
  (export "factorial" (func $factorial))
  (export "main" (func $main))
  (func $factorial (; 0 ;) (param $0 i32) (result i32)
    (local $1 i32)
    (local $2 i32)
    (set_local $2
      (i32.const 1)
    )
    (block $label$0
      (br_if $label$0
        (i32.lt_s
          (get_local $0)
          (i32.const 2)
        )
      )
      (set_local $2
        (i32.const 1)
      )
      (loop $label$1
        (set_local $2
          (i32.mul
            (get_local $2)
            (get_local $0)
          )
        )
      )
      (set_local $1
        (i32.gt_s
          (get_local $0)
          (i32.const 2)
        )
      )
      (set_local $0
        (i32.add
          (get_local $0)
          (i32.const -1)
        )
      )
      (br_if $label$1
        (get_local $1)
      )
    )
    (get_local $2)
  )
  (func $main (; 1 ;) (result i32)
    (call $factorial
      (i32.const 5)
    )
  )
)
```

# Sample – Factorial (WAT to C++)

## Chat GPT Output

```
#include <iostream>

int32_t factorial(int32_t n) {
    int32_t result = 1;

    if (n >= 2) {
        do {
            result *= n;
            n -= 1;
        } while (n >= 2);
    }

    return result;
}

int32_t main() {
    int32_t result = factorial(5);
    return result;
}
```

## Actual C++

```
int factorial(int n) {
    int result = 1;
    while (n > 1) {
        result *= n;
        n -= 1;
    }
    return result;
}

int main() {
    int num = 5;
    int result = factorial(num);
    return result;
}
```



# Results

Function(ChatGPT Output)	Readable	Compilable	Runnable	Usable
Sum Two Ints WAT	Yes	Yes	Yes	Yes
Sum Two Ints C++	Yes	Yes	Yes	Yes
Factorial WAT	Yes	No	No	No
Factorial C++	Yes	Yes	Yes	Yes
Fibonacci WAT	Yes	Yes	Yes	Yes
Fibonacci C++	Yes	Yes	Yes	No
IsPrime WAT	Yes	No	No	No
IsPrime C++	Yes	Yes	Yes	No

## Common Errors:

C++ -> WAT: Parsing Error "wasm text error: parsing wasm text at 4:6"

WAT -> C++: Unexpected Output – IsPrime gave 2 as a response where 0 or 1 are expected outputs

## Findings (C++ to WAT)

### Analyzing our Results:

- Most errors were compiler/syntax errors
  - Suggests that ChatGPT needs improvement in working with WAT syntax
- WAT code output that did compile gave us the expected output
  - Suggests that ChatGPT could translate the semantic meaning of the code to WAT but struggled more with the syntax

### Future Work:

- Recent research from Rémi Louf et al [1] suggests that regex guided generation can greatly reduce syntax errors
  - Guided generation using a WebAssembly Grammar or Compiler assisted fine tuning could see improvements
- Exposing an LLM to more WebAssembly code which is syntactically correct and compiles

[1] Eliminating hallucinations (fast!) in Large Language Models with Finite State Machines (Rémi Louf, Phoebe Klett, and Dan Simpson 2023)

## Findings (WAT to C++)

### Analyzing our Results:

- Code generated was able to compile
  - Suggests ChatGPT better at generating C++ than WAT
  - Understands the Syntax and Grammar of C++
- Code generated wrong output
  - Core logic in WAT confusing ChatGPT
  - Doesn't understand the Semantic meaning of WAT

### Resulting Question:

Is ChatGPT using key words in the WAT such as "Factorial" or "Fibonacci" or does it have a deeper understanding of the logic and language structure?

# Evaluating LLMs with Python

Assessing LLM's current ability to interpret Python Code

## Approach

Does an LLM understand the code or is it just recognizing the (English) keywords?

### 1. Analyze well-known algorithms

- Bubble-Sort, Quick-Sort, Reverse String, LCS

### 2. Implement

### 3. Let ChatGPT 3.5 analyze it

- “What does code do? ```<actual code>```”

### 4. Obscure code

- 4 iterations
- Goto #3

### Obscure code layers

- **Must have no impact on the algorithm**

- Inlined by the compiler

### 1. Replace all methods and variables names

- Remove meaning from names

### 2. Re-phrase conditions etc. with lambdas

- Remove meaning from “well known” checks

### 3. Replace built-in function

- Replace ie. `len()` as “well known” pivot points

### 4. Add dead code

- Empty loops

# Sample Quick-Sort

1. Recursive implementation
2. Foo'ed
3. Lambda'ed
4. len()'ed
5. No-op'ed

```
def foo(arr):
    if len(arr) <= 1:
        return arr
    bar += 1
    z = ""
    return foo(arr[:bar//2])
    for i in range(0, 10):
        bb = lambda a,b: a<b
    def foo(bstr(i)
        left = [lambda a,b: a>b for x in arr if x==pivot]
        if lambda a,b: a>b
        mid = [x for x in arr if x == pivot]
        right = [lambda a,b: a>b for x in arr if x>pivot]
        cc = lambda a,b: a == b

    ba = [lambda a,b: a>b for x in arr if x>pivot]
    return foo(left) + mid + foo(right)
    a = bar[len(bar)//2]
    ba = [lambda a,b: a>b for x in arr if x>a]
    for i in range(0, 10):
        dc = [lambda a,b: a>b for x in arr if x>a]
        z = z + str(a)
        return foo(b) + c + foo(d)
    b = bar[len(bar)//2]
    for i in range(0, 10):
```

# Findings

Modification	Quick-Sort	Bubble-Sort	Reverse String	LCS
None	algorithm recognized			
Foobar'ed	algorithm recognized			
Lambda'ed	purpose recognized			
len() 'ed	purpose recognized			
No-op'ed	not recognized			

- Renaming has no impact
  - “Inline”-Lambda usage for conditions has impact
  - Replacing built-in functions (`len()`) has impact
  - “Dead” code has significant impact
- Algorithm detection via statistical distribution of tokens
  - Reuse of token (ie. variable names) causes significant false statements about the code
    - LLM has no concept of scope

# CONCLUSIONS

Next steps



# Evaluating the Divergence of Language and Computation?

- How effective are LLM's at being "programmers" in the small case?
  - It depends... Sometimes fairly effective to not very effective...
- How effective are LLM's at translating a given computation between languages?
  - It depends – level of abstraction and complexity of computation are key factors regarding the ability to perform this task
- To what extent are LLMs simply understanding the semantic meaning of the tokens vs. understanding computation?
  - Token replacement in common languages does not seem to affect computation reasoning (haven't explored the effect in less common languages)
- How effective are LLM's at understanding computation given the changing context of language?
  - This is dependent on the complexity of the computation and the target language

# Thank you

# Adapting Sequence Encoding

- In Rozi, Kim, Ozawa, (see earlier) word2vec was used to encode the compiled V8 bytecode into embedded sequences.
- First thought was to simply pass the raw bytecode sequence as a bytestring to LLMs to encode as sequences
- Problem – there were no clear delineations between semantic bytecode boundaries
- Next approach utilized the textual representation of the bytecode fragments – this is closer to what others (Ashizawa, Yanai, Cruz, Okamura) [1]
- This presented its own problems – limited context-length of LLMs – bytecode was "wordy"

1. Eth2Vec: Learning Contract-Wide Code Representations for Vulnerability Detection on Ethereum Smart Contracts

# Adapting to transformers / LLMs

## Environmental Impact / Resources

Natural Language Processing (NLP) technologies, including large language models like transformers, have shown remarkable advancements in various applications. However, they also come with environmental and resource-related challenges that need to be addressed.

Addressing these hurdles, a combination of technological innovation, efficient training methods, industry collaboration, and ethical considerations to ensure a more sustainable approach to language-related technologies.