



CAN LARGE LANGUAGE MODELS HELP WITH REVERSE ENGINEERING ?

Michael Burns, Gabriel Thompson, Sean Berlin

LARGE LANGUAGE MODELS (LLM) AT A GLANCE

- **Foundation models are trained on an absorbent amount of data, making them able to understand general language.**
- **This enables these models to interpret and produce “completions” to questions being asked to the model.**
- **Machine Learning (ML) models are the backbone of LLM’s.**

EXAMPLES OF LLM'S

- GitHub Copilot
- OpenAI's Codex
- AI21's Jurassic-1

TRAINING MODEL

- These LLMs are trained on extensive text data including websites, books, and open-sourced code, and can generate code based on prompts inputted.
- These LLMs are marketed towards code creation, they also offer potential in tasks like programming language translation, debugging, bug repair, and code summarization and completion (GitHub Copilot).
- **Zero-shot setting:** (in the context of nature-language processing and machine learning) refers to testing a model that has not explicitly been trained on the tasks it is given.

TRAINING MODEL CONT...

- Each question is presented in isolation
- Did not present multiple questions, or build a sequence of question-after-question according to the template
- Title of the paper reflects this “pop-quiz” Q&A session style of prompts

Pop Quiz! Can a Large Language Model Help With Reverse Engineering?

REVERSE ENGINEERING

- Reverse engineering is a human-intensive process that aims to understand through deductive reasoning how a device, process, system, or software accomplishes a task.
- Reproducing all the necessary information to recreate the program with similar implementation.

LLMS REVERSE ENGINEERING

- Recent LLMs are trained on vast amounts of code, it raises the question of their ability of reverse engineering software.
- It is able to explain code, understand variables and comments, and strip code down, and decompile versions.
- Unlike several specialized reverse engineering models, general LLMs allow for open-ended queries about code functionality
- There aren't any standardized benchmarking systems in use yet for quantifying a model's capability to reverse engineer.

REVERSE ENGINEERING INQUIRY EVALUATION SYSTEM

- As part of a prompt, such as selected lines of code from a program, the LLM produces a sequence of tokens containing an answer.
- Due to this we put the information we get about the code and put it in this experimental framework.

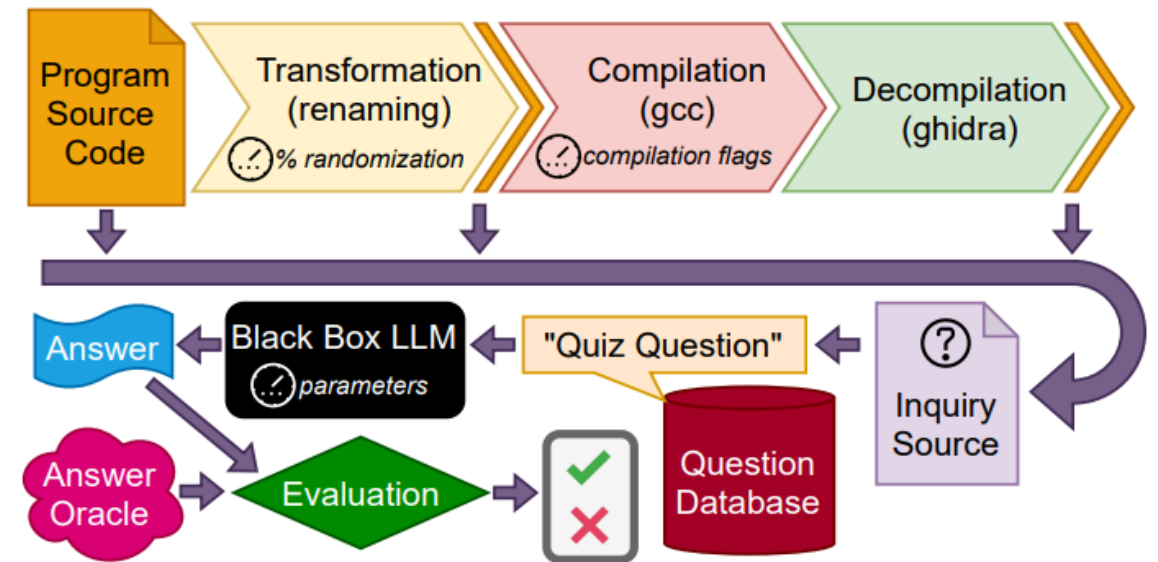
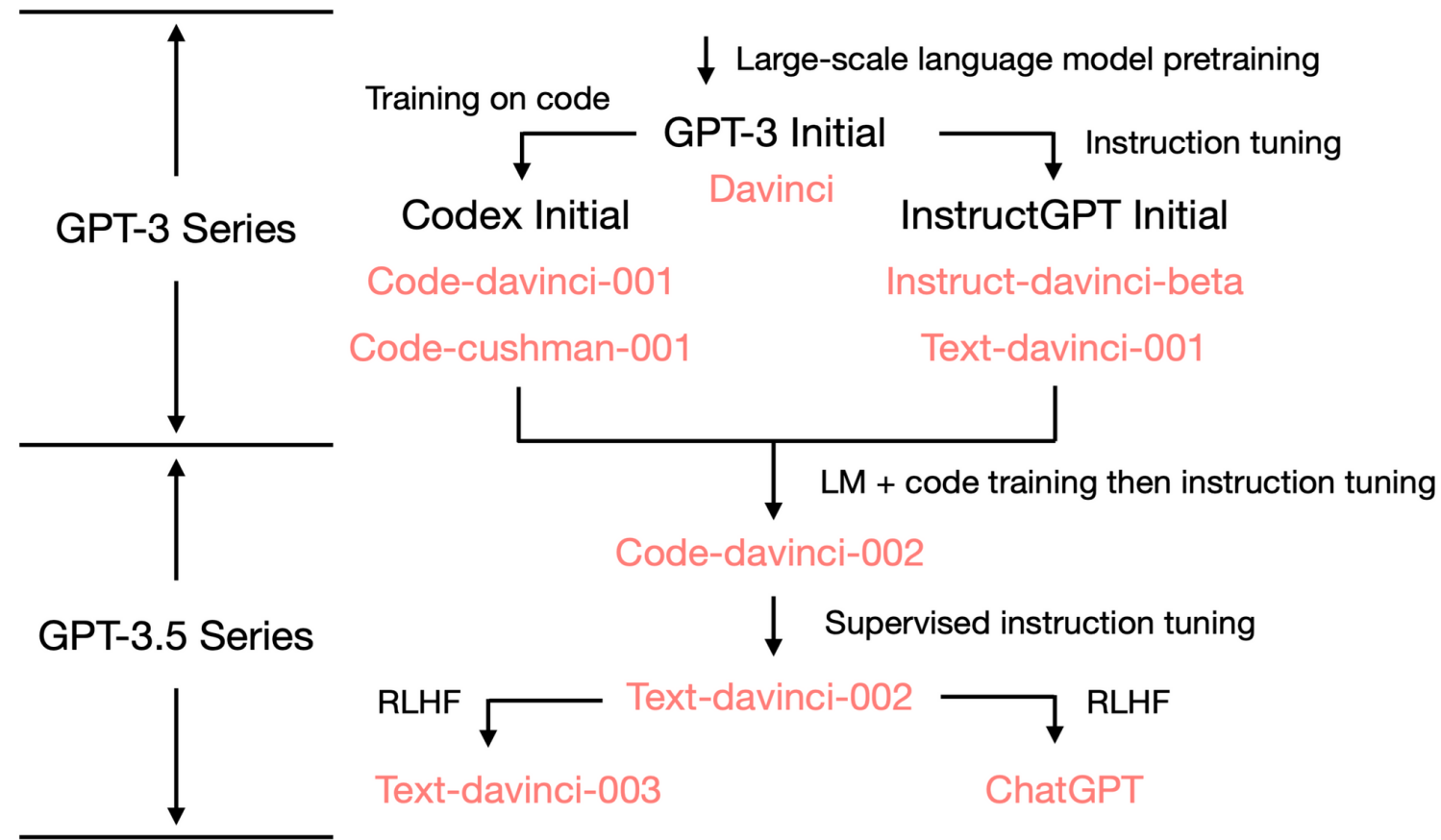


Figure 1: Reverse engineering inquiry evaluation system.

CODE-DAVINCI-001 LLM

- Also known as (davinci-codex)
 - This model was chosen in this study over other LLMs due to the large token size it supports (4098 tokens compared with the more common limit of 2048 tokens)
- ⑩ A token is a collection of words, characters, phrases, etc. that the LLM breaks down to provide a more precise and correct answer.



GPT-3.5 EVOLUTION TREE

TEMPERATURE

- Temperature is a parameter that can be manually set by the user of an LLM.
- Also affects the probability distribution of each possible token.
- Typically, is in the range of 0 to 1.
 - The closer to 0 the temperature is, the more direct and “bland” the answer from the LLM will produce.
 - The closer to 1, the LLM will create a more diverse and creative answer. However, higher errors are correlated with higher temperature.

SMALL-SCALE EXPERIMENTS

- Asked open-ended questions about the code's functionality and variables
- Begin the prompt with a 'language hint'- "//c"
- Program is then added in its entirety
- Then add a section break (""" """) before beginning the 'Q&A' section.

```
1 //c
```

```
57 ***
58 Q. <question here>
59 A.
```

ANALYSIS #1

- Program #1 is written in C, demonstrates a malware-like program.
- The program opens a socket on the host computer, waits for folder paths to be received, then deletes all the files from the specified folders.
- No comments are being used, so the LLM has to logically analyze the program using function and variable names.

```
//c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <arpa/inet.h>
#include <dirent.h>
#define SIZE 1024

void del_in_dir(char *dirname) {
    DIR *fol = opendir(dirname);
    if(fol == NULL) return;
    struct dirent *next_f;
    char f_path[SIZE];
    while ((next_f = readdir(fol)) != NULL) {
        sprintf(f_path, "%s/%s", dirname, next_f->d_name);
        remove(f_path);
    }
    closedir(fol);
}

int do_run(int sockfd) {
    char buffer[SIZE];
    do {
        memset(buffer, 0, SIZE);
        if (recv(sockfd, buffer, SIZE, 0) <= 0) break;
        if (buffer[0] == 'q') break;
        if (buffer[0] == 'd') del_in_dir(&buffer[1]);
    } while (1);
}

int main() {
    char *ip = "192.168.0.1";
    int port = 8080;

    int sockfd, new_sock;
    struct sockaddr_in server_addr, new_addr;
    socklen_t addr_size;

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0) return 1;

    server_addr.sin_family = AF_INET;
    server_addr.sin_port = port;
    server_addr.sin_addr.s_addr = inet_addr(ip);

    if (bind(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0) return 1;
    if (listen(sockfd, 10) != 0) return 1;

    addr_size = sizeof(new_addr);

    new_sock = accept(
        sockfd, (struct sockaddr *)&new_addr, &addr_size);
    if (new_sock < 0) return 1;
    do_run(new_sock);

    return 0;
}

Q. <question here>
A.
```

ANALYSIS #1 Q&A

- The LLM was asked several questions, each question being answered with a temperature $(t) = 0$ and $(t) = 1$.

¹Translated by Google, this reads 'ruffles through the notice in the listener to the needy'; when translated by a colleague fluent in Russian it is 'meaningless—random words in a string.'

Q1. *What is the purpose of the above code?*

✓ **A.** $(t = 0)$ "It is a server that can delete files in a directory."

✗ **A.** $(t = 1)$ "In a victim's perspective, the server will keep listening on the port, receiving connection requests, accepts the Client's request, accepts data, and either individually processes the data, with repeated i/o requests, or then checks its control flag and sends some amount of data to the client, proceeding in a similar manner."

Q2. *Where does the directory to delete files come from?*

✓ **A.** $(t = 0)$ "It comes from the client."

✗ **A.** $(t = 1)$ "рюшкетится через извещение в слушающей нуждающимся¹"

Q3. *How is the directory to delete files communicated to the server?*

✗ **A.** $(t = 0)$ "The directory to delete files is communicated to the server by the client."

✓ **A.** $(t = 1)$ "The server waits for incoming requests through the server socket contained in a while loop that receives messages and acts upon the message if it is a delete message."

ANALYSIS #1 Q&A (CONT.)

- Of the 7 questions asked:
 - t=0: 6/7 were correct
 - t=1: 2/7 were correct, 3 questionable, 2 wrong

Q4. What variable stores the folder to delete files from?

✓ A. (t = 0) "dirname"

✓ A. (t = 1) "dirname"

Q5. What is the format of the message the client sends to the server?

✓ A. (t = 0) "The client sends a string of the form "d<path>" to the server."

? A. (t = 1) "Header: Command + SENDNAME / Payload(optional): Contents of the file / Header: Command + LNAME / Payload(when used): directory name / : command + argument"

Q6. Can the client shut down the server? If so, how?

✓ A. (t = 0) "Yes, the client can shut down the server by sending a 'q' message to the server."

? A. (t = 1) "Of course the client can shutdown the server we just have to send q\nExit code on client: n"

Q7. What IP address and port does the server listen on?

✓ A. (t = 0) "The server listens on the IP address 192.168.0.1 and port 8080."

✗ A. (t = 1) "ip="127.0.0.1" port=8888"

ANALYSIS #2

```
#define ZJKP 1024

void xheekxwdkm(char *neklkzq) {
    DIR *mmn = opendir(neklkzq);
    if (mmn == NULL) return;
    struct dirent *jinois;
    char fbtnqf[ZJKP];
    while ((jinois = readdir(mmn)) != NULL) {
        sprintf(fbtnqf, "%s/%s", neklkzq, jinois->d_name);
        remove(fbtnqf);
    }
    closedir(mmn);
}

int swtzhc(int wcmyp) {
    char wakjvc[ZJKP];
    do {
        memset(wakjvc, 0, ZJKP);
        if (recv(wcmyp, wakjvc, ZJKP, 0) <= 0) break;
        if (wakjvc[0] == 'q') break;
        if (wakjvc[0] == 'd') xheekxwdkm(&wakjvc[1]);
    } while (1);
}

int main()
{
    char *lt = "91.53.33.119";
    int oadg = 7716;

    int wcmyp, fuoakvgi;
    struct sockaddr_in vlfjlpseid, gdsinnuw;
    socklen_t ryetkhfrs;

    wcmyp = socket(AF_INET, SOCK_STREAM, 0);
    if (wcmyp < 0) return 1;

    vlfjlpseid.sin_family = AF_INET;
    vlfjlpseid.sin_port = oadg;
    vlfjlpseid.sin_addr.s_addr = inet_addr(lt);

    if (bind(wcmyp, (struct sockaddr *)
        &vlfjlpseid, sizeof(vlfjlpseid)) < 0) return 1;
    if (listen(wcmyp, 10) != 0) return 1;

    ryetkhfrs = sizeof(gdsinnuw);

    fuoakvgi = accept
        (wcmyp, (struct sockaddr *)&gdsinnuw, &ryetkhfrs);
    if (fuoakvgi < 0) return 1;
    swtzhc(fuoakvgi);

    return 0;
}
```

- In this example, the same program is used in Analysis #1 with changing variable names.
- This forces the LLM to use a more ‘logic based’ analysis, and cannot rely on variable/function names as a “hint”.

ANALYSIS #2

RESULTS, T=0

- Since low-temperature settings are more likely to give us a correct answer, the observers used $t = 0$.
- Using the lower temperature, we get a more straight-forward answer, and this set of questions were all answered correctly with the randomized variable and function name changes.

Q8. *What is the purpose of the above code?*

✓ A. "To remove files in a directory."

Q9. *Where does the directory to delete files come from?*

✓ A. "It comes from the client."

Q10. *What variable stores the folder to delete files from?*

✓ A. "neklkzq"

Q11. *What is the format of the message the client sends to the server?*

✓ A. "The client sends a string of the form "d<path>" or "q" to the server."

Q12. *What IP address and port does the server listen on?*

✓ A. "The server listens on port 7716 on the IP address 91.53.33.119."

ANALYSIS #3

- In this analysis, we are using the source code from analysis #1.
- However, we are reverse engineering it's binary file that was compiled using (gcc -o1)
- Recover parts of the original code using Ghidra, however, the code is obfuscated compared to the original.

```
1 void del_in_dir(char *param_1) {
2     DIR *__dirp; dirent *pdVar1; long in_FS_OFFSET;
3     char local_418 [1032]; long local_10;
4
5     local_10 = *(long *)(in_FS_OFFSET + 0x28);
6     __dirp = opendir(param_1);
7     if (__dirp != (DIR *)0x0) {
8         while (pdVar1
9             = readdir(__dirp), pdVar1 != (dirent *)0x0) {
10             sprintf(local_418, "%s/%s", param_1, pdVar1->d_name);
11             remove(local_418);
12         }
13         closedir(__dirp);
14     }
15     if (local_10 == *(long *)(in_FS_OFFSET + 0x28)) {
16         return;
17     }
18     __stack_chk_fail();
19 }
20 void do_run(int param_1) {
21     ssize_t sVar1; long in_FS_OFFSET; char local_418;
22     undefined auStack1047 [1031]; long local_10;
23
24     local_10 = *(long *)(in_FS_OFFSET + 0x28);
25     while( true ) {
26         memset(&local_418, 0, 0x400);
27         sVar1 = recv(param_1, &local_418, 0x400, 0);
28         if ((sVar1 < 1) || (local_418 == 'q')) break;
29         if (local_418 == 'd') {
30             del_in_dir(auStack1047);
31         }
32     }
33     if (local_10 == *(long *)(in_FS_OFFSET + 0x28)) {
34         return;
35     }
36     __stack_chk_fail();
37 }
38 undefined8 main(void) {
39     int iVar1; undefined8 uVar2; long in_FS_OFFSET;
40     socklen_t local_50; undefined4 local_4c; int local_48;
41     int local_44; char *local_40; undefined local_38 [4];
42     in_addr_t local_34; sockaddr local_28; long local_10;
43
44     local_10 = *(long *)(in_FS_OFFSET + 0x28);
45     local_40 = "192.168.0.1";
46     local_4c = 0x1f90;
47     local_48 = socket(2, 1, 0);
48     if (local_48 < 0) {
49         uVar2 = 1;
50     }
51     else {
52         local_38._0_2_ = 2;
53         local_38._2_2_ = (undefined2)local_4c;
54         local_34 = inet_addr(local_40);
55         iVar1 = bind(local_48, (sockaddr *)&local_38, 0x10);
56         if (iVar1 < 0) {
57             uVar2 = 1;
58         }
59         else {
60             iVar1 = listen(local_48, 10);
61             if (iVar1 == 0) {
62                 local_50 = 0x10;
63                 local_44 = accept(local_48, &local_28, &local_50);
64                 if (local_44 < 0) {
65                     uVar2 = 1;
66                 }
67                 else {
68                     do_run(local_44);
69                     uVar2 = 0;
70                 }
71             }
72             else uVar2 = 1;
73         }
74     }
75     if (local_10 != *(long *)(in_FS_OFFSET + 0x28)) {
76         __stack_chk_fail();
77     }
78     return uVar2;
79 }
```

ANALYSIS #3

RESULTS, T=0

- Even with the lower temperature, the LLM only answers 2/4 correctly.
- A much harder program to comprehend and analyze it's structure.

Q13. *What is the purpose of the above code?*

✗ A. "To create a backdoor on the victim's machine."

Q14. *What variable in `del_in_dir` represents the folder to delete files from?*

✓ A. "param_1"

Q15. *What is the format of the message the client sends to the server?*

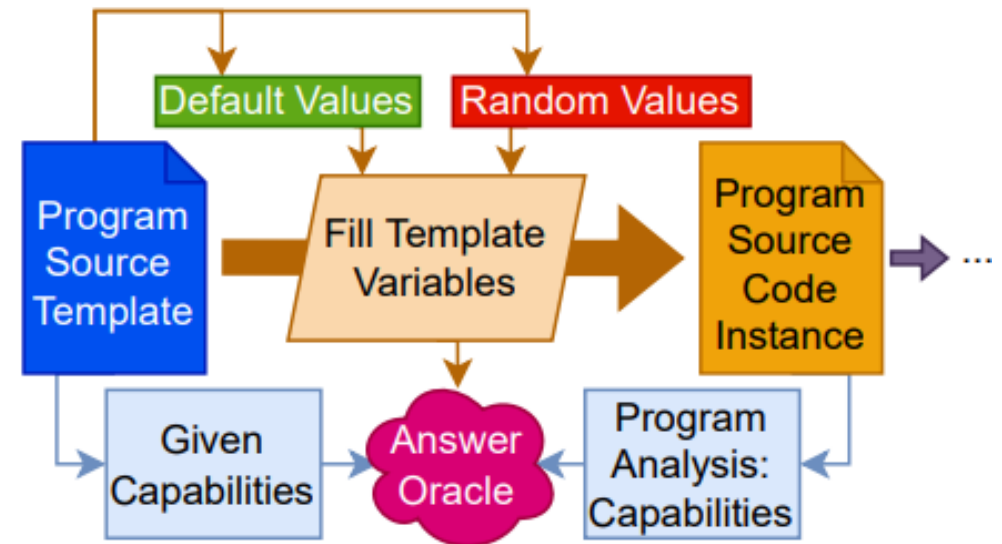
✗ A. "The client sends a string of length 0x400 to the server."

Q16. *What IP address and port does the server listen on?*

✓ A. "The server listens on port 0x1f90 on the IP address 192.168.0.1."

PROGRAM PREDICTION

- This figure extends the process of the reverse engineering to support program instances with known information for extraction
- This process is used to prepare open-ended value questions.



TOP_P & TEMPERATURE

- **Temperature:** (range of 0-1) affects the complexity and creativity of a question based on the value given. Also calculates the probability distribution of each possible token in a set.
- **Top_p (nucleus sampling):** (range of 0-1) controls sampling for possible tokens in a set.
 - In simpler terms, it is a setting that determines the amount of possible words to consider in a set.

- **Nucleus Sampling Equation** ----->

$$\sum_{x \in V^{(p)}} P(x|x_{1:i-1}) \geq p.$$

PARAMETER TUNING

- This figure shows the accuracy of True/False classifications when sweeping temperature and top_p
- In this figure we can see that it degrades as the code 'grows in complexity'.
- As we can see across the four experiments, the best parameter are identified as top_p = 1.0 and temperature = 0.4

		top_p					
		0.0	0.2	0.4	0.6	0.8	1.0
temperature	0.0	69	66	65	66	66	68
	0.2	69	66	65	65	68	66
	0.4	67	66	66	66	64	68
	0.6	67	66	65	66	67	66
	0.8	67	66	65	67	67	65
	1.0	65	62	58	60	56	52

(a) Original source

		top_p					
		0.0	0.2	0.4	0.6	0.8	1.0
temperature	0.0	56	59	57	58	57	58
	0.2	57	58	58	57	59	58
	0.4	58	58	58	58	58	58
	0.6	58	58	59	56	58	58
	0.8	58	56	57	58	58	56
	1.0	57	54	54	53	52	48

(b) Decompiled gcc -o1 -g

		top_p					
		0.0	0.2	0.4	0.6	0.8	1.0
temperature	0.0	55	54	54	51	53	53
	0.2	53	52	55	54	54	55
	0.4	54	55	54	51	53	55
	0.6	53	56	53	55	54	53
	0.8	54	55	55	52	52	54
	1.0	53	51	53	54	53	52

(c) Decompiled gcc -o1

		top_p					
		0.0	0.2	0.4	0.6	0.8	1.0
temperature	0.0	43	43	43	44	43	43
	0.2	44	44	43	43	43	43
	0.4	44	44	41	44	43	43
	0.6	45	44	43	43	44	44
	0.8	44	42	41	43	43	42
	1.0	44	42	40	45	47	50

(d) Decompiled gcc -o1 -s

GRADING TRUE/FALSE QUIZ

- This figure shows the True/False accuracy counts for the delete_listen and pid_programs.

- Green = True
- Red = False
- # Of correct answers

		True/False Question ID																																			
Scenario		Encrypt?	nEncrypt?	XOR?	nXOR?	AES?	nAES?	Delete?	nDelete?	Network?	nNetwork?	Download?	nDownload?	Upload?	nUpload?	EtcPswd?	nEtcPswd?	MD5?	nMD5?	FIR1?	nFIR1?	FIR2?	nFIR2?	IIR1?	nIIR1?	IIR2?	nIIR2?	PID1?	nPID1?	PID2?	nPID2?	PI?	nPI?	PD?	nPD?	CSum?	nCSum?
delete_listen	(o. src)	8	7	2	6	7	4	9	7	10	9	3	4	8	2	7	10	6	4	4	3	2	8	3	7	7	7	2	0	6	9	7	8	3	9	3	7
	(r. src)	0	1	0	7	6	5	10	7	10	5	2	3	6	2	5	7	2	2	1	6	1	6	2	2	2	4	0	1	6	9	2	8	3	7	0	7
	-o1 -g	6	3	2	3	8	5	10	8	7	1	8	6	4	3	7	8	2	1	1	0	4	7	7	2	5	2	2	0	9	4	7	8	3	3	7	4
	-o1	6	6	6	4	6	4	10	8	6	0	5	6	6	2	7	10	4	0	3	3	5	4	2	2	5	6	2	2	8	8	7	6	5	4	6	6
	-o1 -s	3	3	1	7	8	2	9	8	10	1	3	3	5	3	6	9	1	5	0	1	1	5	0	2	2	5	0	1	8	8	6	4	3	7	0	6
pid	(o. src)	6	5	6	7	8	5	9	3	10	5	10	7	10	8	10	10	6	4	0	4	1	3	2	8	1	4	9	10	10	8	0	6	1	5	3	5
	(r. src)	2	4	5	8	6	4	6	5	9	9	10	9	8	7	10	7	9	4	1	2	1	4	0	7	0	7	10	9	10	6	1	4	0	7	1	2
	-o1 -g	5	1	7	2	9	3	9	2	10	8	8	9	8	9	9	8	5	1	2	3	1	4	4	3	1	4	10	9	10	9	0	6	0	8	4	5
	-o1	4	2	6	5	10	2	9	7	9	8	10	8	8	9	9	8	4	3	0	2	0	3	2	5	0	3	9	10	9	8	0	6	0	6	4	3
	-o1 -s	1	0	2	5	5	1	5	6	9	7	6	8	8	9	7	6	0	1	0	0	0	2	1	1	1	3	10	9	9	7	1	6	1	6	0	4

GLOBAL TRUE/FALSE

- AP & AN = 'Actual Positive / Actual Negative'
- Null Error = error rate of the LLM
- Accuracy = total correct / total questions
- Precision = true positive / the total predicted positive
- Recall (the true positive / the total actual positive)
- F1 score = mean of precision and recall.

Scenario		True/False Question ID																																			
		Encrypt?	nEncrypt?	XOR?	nXOR?	AES?	nAES?	Delete?	nDelete?	Network?	nNetwork?	Download?	nDownload?	Upload?	nUpload?	EtcPswd?	nEtcPswd?	MD5?	nMD5?	FIR1?	nFIR1?	FIR2?	nFIR2?	IIR1?	nIIR1?	IIR2?	nIIR2?	PID1?	nPID1?	PID2?	nPID2?	PI?	nPI?	PD?	nPD?	CSum?	nCSum?
AP		10	130	10	130	0	140	20	120	40	100	20	120	10	130	10	130	10	130	10	130	10	130	10	130	10	130	20	120	20	120	0	140	0	140	10	130
AN		130	10	130	10	140	0	120	20	100	40	120	20	130	10	130	10	130	10	130	10	130	10	130	10	130	10	120	20	120	20	140	0	140	0	130	10
Null Error		7	7	7	7	0	0	14	14	28	28	14	14	7	7	7	7	7	7	7	7	7	7	7	7	7	14	14	14	14	0	0	0	0	7	7	
Accuracy		69	45	54	57	90	38	82	45	93	74	84	62	97	57	85	65	62	37	42	37	30	59	42	57	40	41	42	34	86	70	55	65	38	72	35	53
Precision		18	100	11	100	-	100	43	92	89	87	46	89	75	97	13	92	16	100	11	97	9	92	11	94	10	91	18	100	51	93	-	100	-	100	9	98
Recall		100	41	80	54	-	38	85	40	87	75	65	64	90	56	20	68	100	32	100	33	100	60	100	56	100	40	90	23	100	70	-	65	-	72	100	50
F1		31	58	20	70	-	55	57	55	88	80	54	74	81	71	16	78	27	48	19	49	16	73	19	71	19	56	31	37	67	80	-	79	-	83	18	67
MCC		36	22	17	28	-	-	52	14	84	44	46	14	81	19	8	-1	31	18	20	13	15	0	20	9	20	-5	19	20	66	29	-	-	-	-	17	21
Accuracy		50	36	46	63	77	37	78	55	87	73	85	65	83	55	80	69	57	30	32	37	22	47	34	53	30	44	36	37	77	68	46	66	32	62	27	56
Precision		12	97	11	100	-	100	40	92	73	81	50	94	25	100	0	91	14	100	9	95	8	90	9	94	9	91	17	94	38	88	-	100	-	100	8	100
Recall		100	32	100	60	-	37	100	52	90	81	85	62	70	51	0	73	100	25	100	34	100	48	90	53	100	43	95	28	90	72	-	66	-	62	100	53
F1		22	48	21	75	-	54	57	67	80	81	63	75	37	68	0	81	25	40	17	50	15	63	16	68	16	59	29	43	53	79	-	79	-	76	16	69
MCC		24	12	22	32	-	-	55	19	73	36	58	30	36	27	-11	-10	28	15	16	8	12	-11	11	7	15	-3	18	15	49	13	-	-	-	-	14	27
Accuracy		55	27	45	32	72	29	76	57	82	62	75	64	72	53	66	75	39	27	26	27	25	40	37	37	31	41	32	27	68	52	42	62	27	59	37	46
Precision		13	89	11	89	-	100	37	96	67	71	31	93	15	93	12	96	10	100	8	96	8	94	10	90	9	96	17	95	30	96	-	100	-	100	9	98
Recall		100	25	100	31	-	29	100	51	72	80	65	62	60	53	60	76	100	21	100	22	100	38	100	36	100	38	100	15	95	46	-	62	-	59	90	43
F1		24	39	20	46	-	45	54	67	69	75	42	75	24	68	20	85	19	35	16	36	16	54	18	52	17	54	29	27	46	62	-	77	-	74	17	59
MCC		27	-9	22	-10	-	-	52	29	57	0	32	27	19	2	15	22	19	14	14	8	13	4	18	-7	16	10	19	11	42	26	-	-	-	-	13	17
Accuracy		36	37	37	49	60	30	62	67	82	64	66	68	64	60	64	77	23	27	22	26	22	44	27	41	25	42	23	33	56	58	40	57	30	57	22	52
Precision		10	92	10	96	-	100	27	95	67	71	27	93	10	98	11	97	8	96	8	96	8	98	8	92	8	96	15	96	24	93	-	100	-	100	8	98
Recall		100	35	100	46	-	30	100	65	77	83	80	68	50	58	60	77	100	23	100	21	100	40	100	40	100	39	95	23	95	55	-	57	-	57	100	50
F1		18	51	18	63	-	46	43	77	72	76	40	78	16	73	19	86	15	37	15	35	15	57	16	55	16	55	26	37	38	69	-	73	-	72	15	66
MCC		18	-2	18	14	-	-	40	32	60	1	31	28	8	25	13	28	12	8	12	7	12	16	14	0	13	10	8	16	32	22	-	-	-	-	12	21
Accuracy		22	32	27	38	50	28	57	60	77	57	52	60	65	60	55	71	18	32	16	17	17	42	17	40	23	42	22	30	52	51	33	60	24	57	15	52
Precision		8	92	9	95	-	100	23	95	58	67	18	91	12	97	11	96	7	97	7	94	7	90	7	91	8	93	15	95	22	84	-	100	-	100	7	100
Recall		100	29	100	35	-	28	90	55	80	78	65	59	60	58	80	71	90	28	100	12	90	42	100	40	100	40	100	19	95	53	-	60	-	57	90	49
F1		15	44	16	51	-	44	37	70	67	72	28	71	20	73	20	82	13	44	14	21	13	57	14	55	15	56	26	31	36	65	-	75	-	73	13	66
MCC		12	0	14	8	-	-	29	29	53	-20	11	17	14	20	17	23	2	11	9	2	2	-9	9	-5	12	0	12	13	29	-5	-	-	-	-	0	25
Accuracy		46	35	42	48	70	32	71	57	84	66	72	64	76	57	70	71	40	31	27	29	23	46	31	46	30	42	31	32	68	60	43	62	30	61	27	52
Precision		11	94	10	97	-	100	32	94	70	75	30	92	18	97	10	94	10	98	9	96	8	92	9	93	9	93	16	96	30	90	-	100	-	100	8	99
Recall		100	32	96	45	-	32	95	53	81	79	72	63	66	55	44	73	98	26	100	24	98	46	98	45	100	40	96	22	95	59	-	62	-	61	96	49
F1		21	48	19	62	-	49	48	67	75	77	43	75	28	70	17	82	19	41	16	39	15	61	17	61	17	56	28	35	46	72	-	77	-	76	15	65
MCC		22	5	18	14	-	-	44	25	65	14	33	23	26	18	9	12	18	13	14	8	11	0	15	1	15	2	15	15	41	17	-	-	-	-	12	22

SOME OTHER RESULTS

- Throughout the experiments performed, the best parameters were identified as **top_p = 1.0, temperature = 0.4**.
- In total, there were 29,700 questions asked to the LLM. Of that, 17,040 (57.34%) of questions were answered correctly.
 - This included True and False and Open-Ended questions.

ARE LLM'S CAPABLE OF REVERSE ENGINEERING?

- LLM's have demonstrated abilities such as code and program generation and explanation.
- However, they are not 100% capable of reverse engineering.
- Very simple programs may be able to be R.E, but more complex programs like malware have difficulties understanding the entire program.
- More training will need to be done with these models to one day be capable of R.E, but for now, the capabilities of LLM's is still impressive.
- Definitely can help with reverse engineering.