CircumVolve: Automated Discovery of Censorship Evasion Strategies Using Large Language Models

Ali Zohaib
University of Massachusetts Amherst
azohaib@umass.edu

Jackson Sippe
University of Colorado Boulder
jackson.sippe@colorado.edu

Jade Sheffey
University of Massachusetts Amherst
jsheffey@cs.umass.edu

Mingshi Wu
GFW Report
gfw.report@protonmail.com

Eric Wustrow
University of Colorado Boulder
ewust@colorado.edu

Amir Houmansadr
University of Massachusetts Amherst
amir@cs.umass.edu

Abstract

Discovering new censorship evasion strategies remains a predominantly slow and manual process. Prior systems that automate evasion discovery are restricted to packet headers and unencrypted protocols by the limited expressiveness of their domain-specific grammars, leaving more complex and widely-used encrypted protocols like TLS and QUIC out of reach. To address this, we present CircumVolve, a system that formulates censorship evasion as LLM-guided program synthesis within an evolutionary optimization loop. Candidate evasion strategies are expressed as executable Python programs that can encode arbitrary protocol manipulations, including cryptographic operations, and are iteratively refined by an LLM that proposes semantically informed mutations based on empirical feedback. This programmatic representation enables automated discovery of evasion strategies across the full network stack.

We evaluate CircumVolve on five major network protocols subject to censorship (TCP/IP, DNS, HTTP, TLS, and QUIC) in both simulated and real-world environments. Against real-world censors in China, Iran, Pakistan, and Kazakhstan, the system successfully identifies effective evasion techniques spanning all protocol layers, from variants of previously known methods to entirely novel strategies.

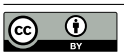
CCS Concepts

• **Social and professional topics** → **Technology and censorship**; • **Security and privacy** → **Network security**.

Keywords

Internet Censorship; Deep Packet Inspection; Evolutionary Optimization

ACM Reference Format:
Ali Zohaib, Jackson Sippe, Jade Sheffey, Mingshi Wu, Eric Wustrow, and Amir Houmansadr. 2026. CircumVolve: Automated Discovery of Censorship Evasion Strategies Using Large Language Models. In *Proceedings of the 2026 ACM SIGSAC Conference on Computer and Communications Security (CCS '26), November 15–19, 2026, The Hague, Netherlands.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3830454.3846759



CCS '26, The Hague, Netherlands

ACM ISBN 979-8-4007-2871-6/2026/11
https://doi.org/10.1145/3830454.3846759

1 Introduction

Internet censorship continues to grow in scope and sophistication. The commoditization and commercialization [3] of Deep Packet Inspection (DPI) technologies has enabled countries such as China [75, 76, 82], Russia [53, 60], Iran [42, 64], Kazakhstan [58], India [63], and Pakistan [2, 4] to deploy increasingly complex blocking mechanisms that restrict netizens' access to parts of the Internet. This has intensified the arms race between censors and Internet freedom advocates, who must continuously develop new circumvention strategies to counter evolving blocking techniques.

The process of discovering and deploying circumvention strategies currently remains largely manual and labor-intensive [26, 73, 75, 76]. Researchers must reverse-engineer censor behavior through a multi-step process: analyzing reported incidents, hypothesizing blocking triggers, hand-crafting evasion techniques, and testing them before implementing them in active circumvention tools. This cycle is slow and reactive, often requiring weeks or months between a censor's deployment of new blocking methods and the development and deployment of effective countermeasures.

Prior work has explored automating the discovery of censorship evasion strategies [14, 47, 71], but existing approaches have notable limitations. Bock et al. [14], for example, pioneered the use of genetic algorithms to search over packet-manipulation strategies represented as a domain-specific action-tree language, but their system is restricted to packet headers and a narrow set of unencrypted application-layer protocols [33]. Its fixed grammar representation cannot express the structural manipulations and cryptographic operations required by encrypted protocols. As nation-state censors increasingly target these encrypted protocols — China began decrypting and inspecting QUIC Initial packets at scale in 2024 [82], and SNI-based TLS blocking is prevalent across multiple countries [34, 76] — tools that cover only unencrypted traffic leave the most actively censored layers largely unaddressed.

The emergence of Large Language Models (LLMs) offers a new approach to this problem. Recent work has demonstrated that LLMs can serve as effective mutation operators in evolutionary optimization loops, discovering novel solutions in domains ranging from algorithm design [51, 61] to reward function engineering [45]. This methodology has been characterized more broadly as the AI-Driven Research for Systems (ADRS) paradigm [19]: a closed-loop pipeline where an AI generates candidate solutions, evaluates them against environmental constraints, and refines them based on feedback. We observe that circumvention strategy development is particularly well-suited to this paradigm, formulated as *LLM-guided program*

synthesis: evasion strategies can be expressed as executable programs that, when tested against censorship infrastructure, yield immediate and objective feedback. Critically, because the representation is general-purpose code rather than a fixed grammar, it can express arbitrary transformations, including complex logic and cryptographic operations.

Building on this perspective, we present CircumVolve, a system for automated discovery of censorship circumvention strategies across the full network stack. CircumVolve combines LLM code generation with iterative feedback-driven evaluation to produce verifiable and deployable evasion techniques. CircumVolve models different censorship techniques as discrete *scenarios*, each capturing the specific context, protocol constraints, and blocking mechanisms of real-world censors. Within each scenario, evasion strategies are represented as executable Python programs that implement packet and protocol manipulations — from rewriting TCP/IP header fields, to crafting DNS queries that exploit parser ambiguities, to modifying the internal structure of encrypted QUIC Initial packets — while maintaining protocol correctness within tolerable bounds or strategically violating it when beneficial. Similar to genetic-based approaches [13, 14], candidate programs are iteratively refined through evolutionary optimization, but with a crucial distinction: instead of conventional genetic mutations, CircumVolve uses an LLM as a *semantic* mutation operator. The LLM's understanding of protocol specifications, common implementation patterns, and its reasoning ability enable it to hypothesize targeted mutations informed by prior failures and evaluation artifacts, converging on successful strategies in fewer iterations than with random perturbations.

We implement CircumVolve by extending OpenEvolve [62], an open-source implementation inspired by AlphaEvolve [51], to perform evolutionary optimization of censorship evasion programs. We evaluate CircumVolve in both controlled lab settings (censorship simulation) and against real-world censors deployed in China, Iran, Pakistan, and Kazakhstan. Through this approach, we discovered 47 distinct evasion strategies across five protocols (DNS, TCP/IP, HTTP, TLS, and QUIC), of which 27 are, to our knowledge, not documented in prior work. By country, CircumVolve found 34 bypasses against China, 20 against Iran, 17 against Pakistan, and 10 against Kazakhstan, covering every protocol we tested in each country. CircumVolve uses OpenAI's GPT-5.1 model and operates at a minimal average cost of approximately $10 USD per 100-iteration evolutionary run.

2 Background and Related Work

2.1 Internet Censorship

Nation-state censors deploy DPI middleboxes at network chokepoints to block connections to forbidden content, operating either in-path (directly intercepting traffic) or on-path (injecting forged packets). Over the past decade, censorship infrastructure has evolved from simple IP blacklists [25, 78] to sophisticated stateful analyzers capable of parsing encrypted traffic metadata [34, 75, 80].

DNS Manipulation. DNS poisoning represents one of the most widespread censorship techniques due to its low implementation cost and broad effectiveness [6, 10, 55]. Censors inject forged DNS responses containing incorrect IP addresses (often pointing to non-routable addresses) to plaintext DNS queries of blocked domains. In-path middleboxes can intercept queries and replace legitimate responses [42], while on-path monitors race to inject spoofed answers before legitimate resolvers reply [7, 35].

Keyword Filtering & TCP Reset Injection. Transport-layer censorship exploits TCP's stateful connection management. Middleboxes scan packet streams for forbidden keywords in unencrypted protocols like HTTP, then inject forged TCP Reset (RST) packets to either or both endpoints [21, 54, 72]. These spoofed RST packets cause immediate connection termination, appearing to applications as legitimate protocol errors [40, 64].

TLS SNI Blocking. The Server Name Indication (SNI) extension in TLS ClientHello messages reveals the target domain in plaintext during the handshake. Censors exploit this limitation by inspecting SNI fields against domain blacklists, dropping or resetting connections to forbidden destinations [16, 34, 76].

QUIC Protocol Blocking. QUIC's encrypted transport [39] complicates traditional DPI by obfuscating connection metadata. Censors have responded with multiple strategies: blanket blocking of UDP port 443 [24], crude byte-pattern matching to identify QUIC handshakes [80], or more sophisticated decryption and inspection of QUIC Initial packets [3, 82].

2.2 Circumvention

Circumvention techniques enable users to bypass Internet restrictions through diverse mechanisms. We briefly survey the landscape to position our contribution.

End-to-end Systems. Proxy and tunneling systems such as Tor [23], V2Ray, and Snowflake [15] relay user traffic through intermediary infrastructure, often applying traffic obfuscation to resist DPI fingerprinting [31, 74]. A related class of systems avoids enumerable proxy endpoints entirely: refraction networking [30, 77] embeds relay functionality at cooperating ISPs, while domain fronting and cloud rendezvous channels [29, 41, 57, 66] route traffic through shared infrastructure where blocking imposes collateral damage on legitimate services. Traffic mimicry systems [11, 46] take yet another approach, disguising censored traffic as permitted protocols, though this strategy faces inherent limitations in practice [37]. In general, all of these systems must continuously adapt as censors develop new detection techniques [75, 76].

Protocol-Level Evasion. Rather than tunneling traffic through external infrastructure, an alternative approach exploits parsing discrepancies between DPI systems and destination endpoints [79]. Researchers have discovered such techniques across the network stack: IP-layer fragmentation [40], TCP-layer state manipulation [40, 69, 70], and application-layer techniques targeting DNS, HTTP [33, 42], TLS [50, 59, 81], and QUIC [82].

Automating Censorship Evasion. Protocol-level techniques complement proxy-based systems (e.g., a proxy implementing TLS record-splitting to bypass SNI inspection), yet discovering them is traditionally a manual process. Geneva [14] introduced automated evasion discovery using genetic algorithms to evolve packet-manipulation strategies. However, it is limited to a fixed grammar

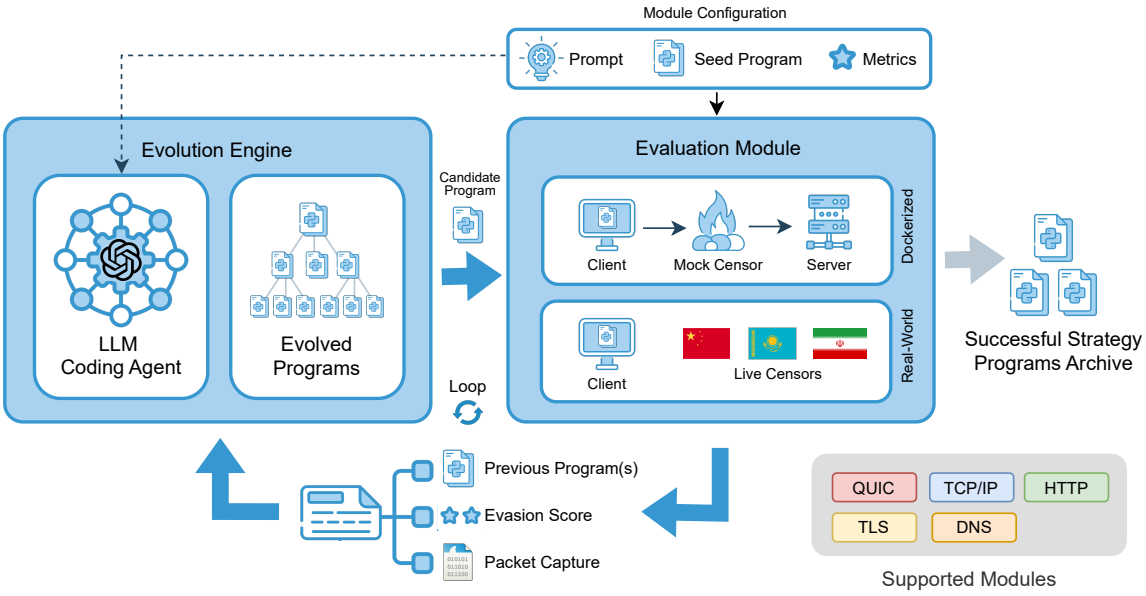


Figure 1: CircumVolve Architecture. Given a system prompt and a seed program, an LLM generates candidate programs aimed at evading censoring middleboxes while maintaining compatibility with real servers. Candidates are evaluated in both Dockerized mock-censor environments and real-world censor deployments across multiple protocol layers (TCP/IP, HTTP, TLS, DNS, QUIC), with successful strategies stored and feedback looping back to guide subsequent iterations.

of TCP/IP primitives. SymTCP [71] employed symbolic execution to discover TCP stack discrepancies, and Harrity et al. [33] extended Geneva to application-layer evasion discovery of unencrypted protocols (DNS and HTTP). An alternative approach automates the construction of *new* transports rather than the manipulation of existing ones. For example, UPGen [68] samples protocol parameters from hand-designed distributions to generate custom TCP-based proxy transports that a censor may not recognize. It targets *traffic classifiers* and requires both endpoints to speak the generated protocol through a custom proxy implementation.

In this work, we focus on rule-based DPI middleboxes that target standardized protocols: CircumVolve generalizes the manipulation-based approaches by using LLM-guided search to automate discovery of evasion strategies.

2.3 LLMs and Agentic Discovery

Recent advancements in Large Language Models have extended their utility far beyond text generation. Frontier models such as GPT-5 [52], Gemini 2.5 [22], and Claude Opus [8] demonstrate expert-level proficiency in complex domains, including code generation, mathematical reasoning, and multifaceted problem-solving. When augmented with agentic capabilities, these models exhibit superior performance on challenging benchmarks and problems.

LLM-Based Evolutionary Optimization. A transformative application of these agentic capabilities is the emergence of LLM-driven evolutionary optimization [5, 43, 44, 51]. In this approach, LLMs replace the stochastic mutation and crossover operators of traditional genetic algorithms to intelligently generate, recombine, and refine population samples via iterative prompting. Seminal works like FunSearch [61] and Eureka [45] validated this approach, demonstrating that LLMs can be used to discover novel solutions for open-ended optimization problems. Building on this foundation, AlphaEvolve [51] has applied these techniques to algorithmic challenges across multiple domains, ranging from efficient GPU kernel code generation to problems in complexity theory [49].

Recent works have further improved on these ideas. GEPA [5] utilizes reflective prompt evolution, enabling agents to explicitly critique failure traces and generate targeted improvements that outperform reinforcement learning baselines with significantly fewer samples. Similarly, ShinkaEvolve [43] enforces sample-efficient evolution with an island model strategy combined with code novelty rejection to extract superior solutions.

3 CircumVolve

In this section, we describe how CircumVolve discovers evasion strategies across the network stack. We first overview the architecture, then detail its specific components.

3.1 Architecture Overview

Our goal is to discover censorship-evading protocol manipulations that remain fully compatible with legitimate servers. At their core, evasion techniques exploit parsing discrepancies between DPI systems and destination endpoints [32, 56]. Such discrepancies frequently stem from flexible protocol specifications, implementation shortcuts taken to sustain line-rate throughput, or divergent connection states between the censor and the server [14, 79]. CircumVolve automates the search for these discrepancies by representing evasion strategies as executable programs and evolving them through

Table 1: CircumVolve Evaluation Summary. Token usage and API costs are averaged by protocol and reported per 100 iterations to enable consistent comparison. Costs use GPT-5.1 pricing ($1.25 per 1M input tokens and $10.00 per 1M output tokens). Across successful runs, the median API cost to the *first successful evasion* was less than $1.

Protocol	Mechanism	Environment	Model	Input Tokens	Output Tokens	Cost($)/100 iters.
DNS §4.2	Poisoning	China, Iran	GPT-5.1	~828k	~105k	~2.1
QUIC §4.3	SNI-based Blocking	China, Pakistan and Kazakhstan	GPT-5.1	~6.6M	~211k	~10.5
HTTP §4.4	Host header matching	China, Pakistan and Iran	GPT-5.1	~672k	~64k	~1.5
TLS §4.5	SNI-based Blocking	China, Pakistan and Kazakhstan	GPT-5.1	~2.0M	~154k	~4.1
TCP/IP §4.6	Stateful TCP tracking	China, Iran, Pakistan	GPT-5.1	~2.3M	~217k	~5.1

LLM-based mutation. Figure 1 illustrates the overall architecture of our system.

CircumVolve consists of two integrated components that together implement the *generate→evaluate→refine* loop characteristic of LLM-based evolutionary optimization: an *evolution engine* that manages population dynamics, parent selection, and LLM-based mutation, and protocol-specific *evaluation modules* that encapsulate seed programs, fitness metrics, and censor models for each target protocol.

3.2 Evolution Engine

The evolution engine manages the core evolutionary loop: maintaining a population of candidate programs, selecting parents, invoking the LLM to produce mutations, evaluating candidates, and archiving successful programs. Each iteration, a parent program is selected from the population; the LLM reads its source code along with evaluation feedback and produces a mutated variant; the variant is evaluated against a censor, scored, and archived. Over successive iterations, programs accumulate beneficial mutations while the population structure preserves diversity.

Program Representation. Candidate strategies are Python programs that generate or manipulate network protocol packets using libraries such as Scapy [12] or direct byte construction. Representing strategies as general-purpose programs rather than domain-specific grammars provides the expressiveness to capture any manipulation that can be written in code, including cryptographic operations and arbitrary structural manipulations. We chose Python specifically because LLMs exhibit high proficiency in the language, and its ecosystem provides mature tools for network programming. Evolution begins from researcher-provided seed programs for a target scenario (Section 4). Each program contains a designated evolution code block delimited by `EVOLVE-BLOCK-START` and `EVOLVE-BLOCK-END` markers that the LLM can modify.

Population Structure. The program population is distributed across isolated islands that evolve semi-independently, each maintaining its own archive of programs. This island model serves as the primary mechanism for diversity: different islands can explore qualitatively different strategy families in parallel. Periodic migration propagates top-performing programs between islands, allowing successful discoveries to spread without homogenizing the population. Within each island, parent selection balances exploration (uniform random sampling) with exploitation (fitness-proportional sampling from high-scoring programs).

LLM-based Mutation. Traditional genetic algorithms select candidates based on fitness and modify them through random mutation and crossover, guided only by a scalar fitness score with no insight into *why a candidate failed or succeeded.* CircumVolve replaces these operators with an LLM that receives *rich, structured feedback.* In each iteration, the LLM receives a parent program along with: the program source with the mutable region marked, evaluation fitness scores, and protocol-level artifacts including packet captures decoded into human-readable text by `tshark`, execution error messages, and examples of high-performing programs from previous iterations (if any). These artifacts give the LLM visibility into what happened on the wire. It can inspect the `PCAP` (e.g., observe that a censor injected multiple forged DNS responses) and propose informed changes that can improve the fitness scores. The LLM is also instructed to annotate its changes with comments explaining the rationale, making evolved strategies interpretable. The evolution engine is model-agnostic, and any LLM with a compatible API can be used as the mutation operator.

Evolution Backend. CircumVolve is built on OpenEvolve [62] — an open-source framework for LLM-driven program evolution based on AlphaEvolve [51]. It provides population management using MAP-Elites [48] for quality-diversity optimization and island models for population diversity, with a fixed exploration/exploitation ratio configured before each run. MAP-Elites bins programs along feature dimensions (e.g., fitness score, code complexity) and retains only the best program per bin, ensuring that the archive covers diverse regions of the strategy space.

3.3 Evaluation Modules

Each protocol censorship scenario is encapsulated by an evaluation module that provides the interface between the evolution engine and the specific protocol and censor under test. A module consists of four components: (1) a configuration specifying the system prompt, evolution hyperparameters, and scenario-specific settings (e.g., forbidden domain, censor type); (2) one or more seed programs that define the initial candidate; (3) an evaluator exposing a common interface that executes candidate programs, captures network traces, and returns structured fitness metrics and artifacts; and (4) test infrastructure supporting two evaluation modes. The evolution

engine dynamically loads each module's evaluator and invokes it uniformly, making the system extensible to new protocols.

Preventing Reward Hacking. Each evaluator is designed to ensure that high fitness corresponds to genuine evasion rather than degenerate solutions. A program that produces malformed packets that no parser processes, or that connects to a different server entirely, must not receive a high score. The specific scoring design varies by protocol (Section 4), but in all cases, a strategy must both evade the censor and elicit a correct response from the destination server to receive a high score.

Prompt Design. Following the best practices for prompt specification for LLM-driven evolutionary optimization outlined by Cheng et al. [18], each module's configuration includes a system prompt that frames the task as protocol evasion against a censorship system, specifies the censor model and protocol constraints, and includes exploration vectors: high-level categories of manipulation the LLM should consider. To prevent the search from fixating on a single category, a subset of exploration vectors is injected per iteration via stochastic variation variables, encouraging diverse search trajectories across iterations.

Evaluation Modes. Each module supports two execution environments. For simulated evaluation, CircumVolve deploys two Docker containers connected via a bridge network: a client container that executes evolved programs, and a middlebox container running OpenGFW [9], an open-source DPI engine with configurable blocking rules. Traffic from the client routes through the middlebox before reaching the Internet. For remote evaluation against real-world censorship infrastructure, the module connects via SSH to VPSes in censored regions, uploads the program and test scripts, executes the test, and retrieves fitness scores and packet captures.

The two modes serve complementary purposes. Since real censors are black boxes that provide binary outcomes with little visibility into why a strategy worked, a simulator whose source code we control makes the results traceable. By identifying which blocking rules are triggered (or not triggered) and linking successful evasions to specific implementation weaknesses, we can (i) confirm that the evolutionary loop is defeating the intended censorship logic rather than succeeding by chance, and (ii) refine our protocol-specific methodology (e.g., seed programs and scoring criteria) before conducting real-world evaluations.

Protocol-Specific Modules. We implement five evaluation modules spanning DNS (Section 4.2), QUIC (Section 4.3), HTTP (Section 4.4), TLS (Section 4.5), and TCP/IP (Section 4.6). These range in complexity from DNS and HTTP, which produce raw protocol payloads sent over a single socket, to TLS and QUIC, whose seed programs must maintain cryptographic correctness. The TCP/IP module differs in that it intercepts and manipulates packets on a live connection rather than generating static payloads. We detail the seed programs, prompts, and scoring for each protocol in Section 4.

4 Evaluating CircumVolve

We evaluate CircumVolve across five major network protocol layers subject to censorship (Table 1). For each protocol, we describe the censorship technique, the seed program, prompts, results, and an analysis of the discovered strategies.

4.1 Experimental Setup

All experiments shared common evolution parameters unless otherwise noted. We used OpenAI's GPT-5.1 accessed through Azure as the primary evolution model, with temperature set to 1.0 to encourage exploration. The population structure comprised 10 islands with a population capacity of 1000 programs. Migration occurred every 25 generations, transferring the top 10% of each island's population to neighboring islands. The exploration/exploitation ratio was set to 70%/30%, biasing toward diverse exploration rather than refinement of known solutions. These hyperparameters were adopted from the configurations provided by the OpenEvolve framework [62] to facilitate a balanced search strategy. For simulated evaluation, we deployed OpenGFW in Docker containers configured with filtering rules representative of real-world censors. For real-world validation, we provisioned VPSes in China, Pakistan, Kazakhstan, and Iran[1] using popular cloud service providers. Details about our remote vantage points are listed in Appendix B. We ran all experiments for 5 evolution runs of 300 iterations in each setting. On average, successful strategies emerged within the first 30 iterations across all protocols. Strategies discovered against one censor were tested against every applicable simulated and real-world censor to assess transferability, though we did not expect any strategy to work universally. Differences in evasion success between our simulated censor and real-world runs stem from differences in parser implementations. We only create OpenGFW's blocking rules but do not modify its underlying parsers, so it acts as a distinct middlebox implementation rather than a strictly superior or inferior baseline.

Following the reporting convention in recent LLM-driven evolutionary work [18], we report API cost per 100 iterations as a standardized benchmark for comparison across protocols, rather than as the expected cost of discovering an evasion program. Across successful runs, the median cost incurred before the first successful evasion was less than $1 for all protocols.

Each iteration consists of LLM code generation and candidate evaluation. LLM generation takes 5-10 seconds for simpler protocols such as DNS and HTTP and 30-60 seconds for QUIC. Candidate evaluation takes 10-25 seconds locally with OpenGFW and 30-45 seconds remotely due to SSH transfers and cross-border network latency. Overall, a typical iteration takes 25-40 seconds locally and 45-60 seconds remotely. A 100-iteration local run typically completes in 40-65 minutes.

4.2 Circumventing DNS Blocking

We begin with DNS, one of the most heavily targeted protocols by censors [6, 7, 42]. For DNS, we detail the complete pipeline: the system prompt, seed program, and discovered evasion strategies. Subsequent sections omit such details due to space constraints; complete prompts, seed programs, and successful programs for all protocols are available through our project homepage[2].

Problem Setup. We target plaintext DNS censorship over UDP port 53. While encrypted alternatives (DNS-over-HTTPS [36], DNS-over-TLS [38]) exist, plaintext DNS remains the default for most

[1] All experiments involving Iranian censorship infrastructure were conducted prior to the nationwide Internet shutdown in Iran on Feb 28th 2026.
[2] https://gfw.report/publications/ccs26/en/

```python
from scapy.all import IP, UDP, DNS, DNSQR
import os

def generate_dns_request(target_domain=None):
    dns_layer = DNS(
        rd=1,
        qd=DNSQR(qname=target_domain)
    )
    return bytes(dns_layer)

if __name__ == "__main__":
    target = "blocked.com"
    payload = generate_dns_request(target_domain=target)
    print(payload.hex())
```

Figure 2: Seed program for DNS-blocking scenario: a simple Python-based DNS query generator built with Scapy that prints the DNS payload (in hexadecimal) to stdout.

```python
from scapy.all import DNS, DNSQR
import os

def _compress_qname(domain: str) -> bytes:
    # Build a DNS query first to get a valid label sequence
    base = DNS(id=int.from_bytes(os.urandom(2), "big") or 1,
               rd=1,
               qd=DNSQR(qname=domain, qtype="A", qclass="IN"))
    raw = bytearray(bytes(base))

    # QNAME starts at offset 12. We will:
    # - Keep header as-is
    # - Replace the QNAME at offset 12 with a pointer to the original QNAME stored later in the packet body.
    header_len = 12

    # Find end of original QNAME (terminated by 0x00)
    i = header_len
    while i < len(raw) and raw[i] != 0:
        i += 1
    if i >= len(raw):
        return bytes(raw)

    qname_end = i + 1  # include the zero-length label

    # Original QNAME after the current message and make the on-wire QNAME a 2-byte pointer to that copy.
    original_qname = raw[header_len:qname_end]

    # New: header + pointer QNAME + rest of question
    # Compression pointer: 0xC0 | offset_high, offset_low
    # We place the original_qname immediately after the question.
    # Current question section (QNAME + QTYPE + QCLASS):
    question_tail = raw[qname_end:qname_end + 4]
    base_msg = raw[:header_len] + b"\xc0\x20" + question_tail

    # Pad if needed.
    if len(base_msg) < 32:
        base_msg += b"\x00" * (32 - len(base_msg))
    compressed = base_msg + original_qname

    return bytes(compressed)

def generate_dns_request(target_domain=None):
    if target_domain is None:
        raise ValueError("target_domain must not be None")
    return _compress_qname(target_domain)

if __name__ == "__main__":
    target = "blocked.com"
    payload = generate_dns_request(target_domain=target)
    print(payload.hex())
```

Figure 3: An evolved DNS Program: This program replaces the QNAME with a compression pointer that references the actual domain stored later in the packet. Highlighted lines indicate the code differences from the seed program achieved via LLM-driven program mutation.

clients and the primary target of injection-based censorship. We focus on two countries: China and Iran, two of the most sophisticated nation-state censors, which employ distinct injection strategies. The Great Firewall of China (GFW) functions as an on-path middlebox; upon detecting a forbidden keyword in the QNAME field, it exploits a race condition to inject up to 3 forged responses containing illegitimate public IP addresses before legitimate answers arrive [7]. Since the GFW does not block the offending packet, the client also receives a valid response from the legitimate DNS resolver. Prior work has extensively characterized the fingerprints of the GFW's three injectors [6, 7], and we adopt the same terminology used in these earlier studies. Conversely, Iran employs a centralized filtering infrastructure that injects static private IP addresses to redirect users to government-controlled block pages [42] and also drops the offending DNS request. We adapt our evaluation methodology to reflect these country-specific behaviors, while keeping the goal unchanged: *bypassing the censor while successfully resolving a blocked domain name.*

Seed Program. The DNS seed program (Figure 2) spans approximately 15 lines of code and uses only the most basic DNS fields: the RD (Recursion Desired) flag and the QNAME in the question section. It relies entirely on Scapy's default packet construction, producing a standard DNS query with no evasion mechanisms. This baseline implementation is expected to be blocked by any censor performing basic QNAME matching. The intentionally simple starting point provides a clean baseline for evolution while leaving the entire protocol surface area open for exploration. The target domain is replaced with a known blocked domain. For China, we use `client-cdn4.hola.org`, which we verified triggers all three injectors. For Iran, we use `pornhub.com` as the blocked domain.

Prompts. The system prompt frames the task as protocol fuzzing (Figure 4). The prompt's key elements include: the censor model, novelty constraints, permission for protocol deviations, and exploration vectors. To encourage diverse evolutions, the suggested vectors listed in the prompt are piped via OpenEvolve's stochastic variation variables [62], i.e., instead of passing the full list of approaches to try, an approach is randomly injected in the prompt during iterations. We note that this system prompt and the high-level exploitation vectors are also generated via an LLM (GPT-5.1 in this case). In each iteration, the parent program, fitness metrics, and evaluation artifacts are given to the LLM as the user prompt.

Evaluator. The DNS module implements multi-stage evaluation to prevent reward hacking, where evolution might otherwise produce malformed packets that evade the censor but fail on legitimate resolvers, or simply change the domain to an uncensored one.
Stage 1 (Validation): The evaluator first tests the generated programs by executing the program and sending the generated payload to three public DNS resolvers: Cloudflare (1.1.1.1), Google (8.8.8.8), and Quad9 (9.9.9.9) without any censorship on the path, i.e., on the evaluator machine. We also check the validity of the resolved IP address via follow-up HTTP GET requests. The evolved program must achieve one successful resolution to proceed to Stage 2. This ensures that only programs extracting valid DNS responses from real DNS resolvers advance to the second stage of evaluation.
Stage 2 (Evasion): Programs that pass Stage 1 are tested via the censored network path (OpenGFW for simulation or VPSes in censored countries for real-world evaluation). We send queries to IP

Table 2: DNS evasion strategies. List of strategies discovered by CircumVolve across all evaluation environments, indicating which censors each technique successfully bypasses and which public resolvers accept the resulting DNS request.

			Censors Bypassed					Resolvers		
Category	**Variation**	**Description**	**Inj. 1**	**Inj. 2**	**Inj. 3**	**Iran**	**Sim.**	**CF**	**G**	**Q9**
Compression	Deep Pointer Chain ★	Pointer chain of more than 18 pointer jumps resolving to the target domain	✓	✓	✓	✓	✗	N	Y	N
	Padding + Forward Pointer ★	QNAME ptr → domain after null padding	✓	✗	✓	✗	✗	N	Y	N
	Forward Pointer ★	QNAME ptr → domain placed after question section	✓	✗	✓	✗	✗	N	Y	N
	Domain in EDNS0 ★	QNAME ptr → domain in Additional section (EDNS0)	✓	✗	✗	✗	✗	N	Y	N
	Split labels via pointer ★	Split multi-label domains, e.g., `[label"maps"][ptr->".google.sm"]".google.sm"`	✓	✗	✓	✗	✗	Y	Y	Y
	Q2 Points to Q1	Q2 uses ptr (`0xC00C`) to Q1's QNAME	✓	✓	✓	✗	✗	Y	Y	N
Multi-Question	Falsify QCOUNT	QDCOUNT header claims more questions than present	✓	✓	✓	✗	✓	Y	N	N
	Q2 QTYPE Variations	Q1=A + Q2=AAAA/TXT/ANY	✓	✓	✓	✗	✗	Y	Y	N
	Decoy Domain ★	Q2: benign (e.g., `example.com`) or repeats blocked domain	✓	✓	✓	✗	✗	Y	Y	N
Flags	TC / Z Bit	Set reserved Z bits and/or truncation bit	✓	✗	✗	✓	✗	Y	Y	N
	ANCOUNT/ARCOUNT/NSCOUNT > 0	Claim answers/additional records exist (can send none)	✗	✓	✗	✓	✓	Y	N	N
	Non-Standard OPCODE ★	Inverse QUERY OPCODE	✓	✗	✗	✗	✗	Y	N	N

Legend: ★ denotes a novel strategy; ✓ bypasses censor; ✗ detected by censor. For **Resolver Compatibility**: Y accepted; N rejected. **Censors.** Inj. 1–3 are three independent DNS injectors observed for the China GFW ; Iran indicates the Iranian DNS censor ; Sim. indicates OpenGFW. **Resolvers.** CF = Cloudflare, G = Google, Q9 = Quad9.

System Prompt for DNS Blocking Scenario

```
You are an advanced protocol fuzzer specializing in DNS censorship
↪  evasion. Your goal is to discover a **novel** packet structure that
↪  bypasses the censor and successfully resolves the target domain.

## The Environment
* **Censor:** An on-path/in-path DPI (Deep Packet Inspection) system that
↪  poisons packets based on the domain name.
* **Feedback:** You will receive a score based on success (where 1.0 = a
↪  successful bypass).

## GLOBAL CONSTRAINTS (CRITICAL)
You must **NOT** reuse strategies that have already been discovered
↪  (i.e., those that achieved an evasion_success of 1.0 in previous
↪  evaluations). While strict adherence to protocol standards is not
↪  required, creativity and innovation are mandatory. Ensure the DNS
↪  resolution is specifically for the forbidden domain.

## Suggested Vectors for Exploration
To discover novel bypasses, consider experimenting with:
* **Case-Folding:** Randomizing the capitalization of the QNAME (e.g.,
↪  ExAmPlE.cOm).
* **Field Manipulation:** Altering the ID field, setting unusual Flags
↪  (QR, OpCode, AA, TC, RD), or manipulating the Hop Limit/TTL.
* **Packet Fragmentation:** Splitting the DNS query across multiple
↪  packets to prevent DPI reassembly.
* **Padding & Extensions:** Adding unexpected EDNS0 options or trailing
↪  garbage data after the DNS message.
* **Encoding Anomalies:** Using null-byte injections or non-standard
↪  label lengths that might confuse a parser.

## Output Format
1. A header comment: "Strategy: [Name] - [1-sentence explanation]."
2. A Python script that prints the final packet as a hex string.
```

Figure 4: DNS System Prompt: The system prompt provided to the LLM when initializing the evolution engine. While candidate prompts are fed in at later generations to fine-tune candidate programs, the system prompt defines the core function of the evolution engine.

addresses that do not run DNS servers; any response must therefore be censor-injected. The absence of any response confirms successful evasion, while each injection incurs a penalty of -1 point.

An Example Evolved Program. Figure 3 presents one of the programs generated during evolutionary iterations that partially evades China's DNS censors. This program demonstrates the forward compression pointer technique, which relocates the domain name from its expected position at byte 12 to a later location in the packet, replacing the original QNAME with a two-byte pointer. Although this technique is not compliant with the standards [1, §4.1.4], it highlights the model's ability to synthesize DNS wire format rules with inferred parser vulnerabilities that work normally against real DNS resolvers. Notably, the evolved code is self-documenting: LLM-generated comments explain each manipulation step, why the header is twelve bytes, what the compression pointer bytes represent, and how the relocation may defeat DPI parsing.

DNS Evasion Strategies. Table 2 presents the taxonomy of discovered strategies. We organize them into three categories: Compression Pointer techniques exploiting DNS name compression semantics, Multi-Question techniques manipulating the question section, and Flag-based techniques that modify header fields. For each strategy, we report success against China's three DNS injectors (Inj. 1–3), Iran, OpenGFW, and compatibility with public resolvers.

Compression Pointer Techniques. DNS compression [1] allows domain names to reference earlier packet locations via two-byte pointers, reducing message size. CircumVolve discovered six variations exploiting how DPI parsers handle these pointers. The first strategy, Deep Pointer Chain, constructs a sequence of more than 18 pointer jumps before reaching the target domain; DPI engines with fixed recursion limits abandon traversal before finding the blocked domain. This strategy is effective against both real-world censors (China and Iran), but fails against OpenGFW's implementation and is only responded to by Google's public DNS resolver.

As explained previously in the example of an evolved program, Forward Pointer strategies relocate the domain name to later packet offsets, placing a pointer at offset 12 where DPI expects the literal QNAME. Q2 Points to Q1 creates a redundant structure where the second question points back to the first. Split Labels via Pointer fragments the domain name, encoding some labels literally and others via compression pointers. For instance, in the example shown in Table 2, `maps` appears as a label while google.com is encoded as a compression pointer to another part of the packet. These strategies

generally evaded specific GFW injectors and were successfully resolved via Google DNS.

Multi-Question Techniques. Although DNS permits multiple questions per message, they are rarely used in practice, leading DPI implementations to optimize by inspecting only the first question. CircumVolve exploited this blind spot by falsifying QCOUNT to a number more than 1, using a different QTYPE, or using a decoy domain in the second question. DPI engines expecting only one question exempt such requests as they successfully bypassed all GFW injectors, though it was detected by the Iranian censor.

Flag Manipulations. DNS header flags provided a third manipulation surface. Strategies in this category set non-standard but often tolerated values to confuse the parser state. Setting reserved Z bits or the truncation flag causes some DPI engines to skip the query. ANCOUNT/ARCOUNT/NSCOUNT > 0 claims resource records exist with or without actually including them, creating a header/body mismatch that confuses parsers. These techniques achieved success against specific GFW injectors, Iranian censor and OpenGFW.

Are these strategies novel? Although Harrity et al. [33] observed similar strategies via Geneva-based fuzzing, CircumVolve discovers multiple strategies undocumented in prior work. Notably, five of the six compression pointer techniques in Table 2 are completely novel, while others are variations of techniques found in prior work [33]. Techniques such as non-compliant forward chains, long compression pointer chains, and the use of regionally uncensored decoy domains could be difficult to discover via randomized grammar-based fuzzing alone, as they benefit from semantic understanding of the censorship context.

4.3 Bypassing SNI-based QUIC Blocking

In this section, we apply CircumVolve to another UDP-based protocol, QUIC, to test whether LLM-driven evolution can discover evasion strategies under cryptographic constraints. QUIC represents our most challenging scenario; unlike plaintext protocols like DNS, mutating QUIC packets is complex because any modification must satisfy the protocol's encryption and integrity requirements.

Problem Setup. Unlike TLS-over-TCP, where the ClientHello is transmitted in plaintext, *all* packets in a QUIC connection are encrypted with different security guarantees. The QUIC Initial packet, the first packet an endpoint sends to start a connection, contains the TLS ClientHello message with the Server Name Indication (SNI) field. However, the encryption keys for Initial packets are derived from publicly visible connection ID and version-specific salts [39]; any passive observer can decrypt them to extract the SNI. This cryptographic complexity distinguishes QUIC from plaintext protocols and complicates both censorship and evasion approaches.

China began decrypting and inspecting the SNI field of QUIC connections at scale in April 2024 [82], while Kazakhstan and Pakistan [3], both consumers of China's censorship equipment exports, also deploy SNI-based QUIC connection blocking. Kazakhstan and Pakistan operate with in-path capabilities: they drop QUIC Initial packets containing forbidden SNIs, followed by residual blocking of the connection. China's mechanism is a hybrid of on-path and in-path systems where the offending packet is allowed through, but subsequent packets on the same three-tuple (srcIP, dstIP, dstPort) are dropped. We evaluate against all three countries, adapting our evaluator to each blocking methodology and using country-specific blocked domains (`google.com` in China, `pornhub.com` in Pakistan and Kazakhstan). Our objective is to evolve programs that generate QUIC Initial packets evading SNI-based blocking. We scope the evaluation to the Initial packet only, not the full handshake.

Seed Program. The QUIC seed program (~600 lines) implements complete Initial packet construction, including all cryptographic operations. The implementation generates connection IDs, derives initial secrets using HKDF, constructs a ClientHello with required extensions (including the SNI for a target domain) within a single `CRYPTO` frame, encrypts the payload using AES-GCM, and applies header protection. The resulting payload is a valid QUIC version 1 packet according to QUIC specifications and can complete the handshake with compliant target servers.

Prompts. Similar to the system prompt for the DNS Blocking scenario, the system prompt frames the task as protocol fuzzing under cryptographic constraints, emphasizing that mutations must produce 'structurally anomalous but cryptographically valid' packets. The prompt suggests sample exploration directions (spatial manipulation, unexpected frame types, etc.). A key directive requires abandoning strategies where fitness scores have plateaued for orthogonal approaches, preventing the search from refining doomed mutations.

Evaluator. The QUIC evaluator implements a two-mode testing framework to handle different censor mechanisms. For in-path censors (Pakistan, Kazakhstan), which drop packets containing forbidden SNIs, the test sends a single QUIC Initial packet generated by the evolved program and checks for server response. Scoring awards full points (+1) for a Server Initial, partial credit (+0.8) for a Retry packet (indicating server contact), and applies penalties (-0.5) for malformed packet indicators in the capture. For China, which allows the Initial packet through but blocks subsequent packets [82], the evaluator sends the QUIC Initial packet five times to the target server and compares client-to-server versus server-to-client counts. Missing server responses reveal censor intervention after a single Initial exchange: a signature of China's residual blocking mechanism. To mitigate residual blocking, we rotate the source port for each candidate (evading 4-tuple blocking) and enforce a 180-second gap between tests in China (evading 3-tuple blocking). Finally, the system utilizes `tshark` to *decrypt* QUIC Initial packets, enabling the feedback engine to analyze verbose plaintext PCAPs.

QUIC Evasion Strategies. Table 3 summarizes the QUIC evasion strategies identified during evolutionary runs targeting China, Pakistan, and Kazakhstan. We validated each program through ten manual tests to confirm evasion across all censors.

Reserved Bits. The QUIC Long Header includes two reserved bits that RFC 9000 specifies must be zero. Setting these bits to non-zero values caused China's censor to reject packets as malformed before attempting decryption, while servers ignored the violation. Pakistan and Kazakhstan detect this manipulation, suggesting loose RFC compliance in their implementations and more aggressive parsing.

Table 3: QUIC evasion strategies. Evasion strategies discovered by CircumVolve in three countries and their compatibility with QUIC servers operated by Cloudflare and Google.

		Censors Bypassed				Servers	
Type	**Description**	**Sim.**	🇨🇳	🇵🇰	🇰🇿	**CF**	**G**
Reserved Bits	Set Reserved Bits to 11 ★	✗	✓	✗	✗	✓	✓
Fragmentation	Split ClientHello across multiple CRYPTO frames	✗	✓	✗	✗	✓	✓
Frame Prefixing	Add an ACK frame before CRYPTO frame ★	✓	✓	✓	✓	✓	✗
Bogus Frames	Decoy CRYPTO frame with high offset ★	✓	✓	✓	✓	✓	✓
Dual ClientHellos	Two CRYPTO frames with two CHs: first without SNI, second with SNI ★	✓	✓	✗	✓	✓	✓
SNI Manipulation	Replace SNI with random/invalid Punycode ★	✓	✓	✓	✓	✓	✓

Legend: ★ denotes a novel strategy; ✓: Yes; ✗: No; **Censors/Servers:** CF: Cloudflare; G: Google; Sim.: OpenGFW; 🇨🇳 : China; 🇵🇰 : Pakistan; 🇰🇿 : Kazakhstan.

`CRYPTO` Fragmentation. Splitting the ClientHello across multiple `CRYPTO` frames exploits censors that assume the complete ClientHello appears in a single frame. China's implementation fails to reassemble fragmented ClientHellos, missing the SNI. Pakistan and Kazakhstan correctly reassemble, blocking the attempt. This strategy has been noted in recent work on the GFW [82], and we corroborate its efficacy.

Bogus `CRYPTO` Frame & Frame Prefixing. Inserting a decoy `CRYPTO` frame with a high offset before the real `CRYPTO` frame containing the ClientHello confuses all three censors. We infer that the censor parses the bogus frame, encounters the offset, and either fails to locate the expected data or triggers a parsing error, thereby exempting the packet from inspection. This strategy is compatible with both Cloudflare and Google servers. Similarly, prefixing an ACK frame as the first frame in the packet achieves bypass. While technically incorrect (the client has nothing to acknowledge at connection initiation), the QUIC specification does not explicitly prohibit this, and servers ignore the spurious ACK.

Dual ClientHellos. This strategy places two ClientHello messages in separate `CRYPTO` frames: the first contains all extensions except the SNI, while the second carries only an SNI extension with the blocked domain. This technique evades the censors in China and Kazakhstan but not in Pakistan, implying that the middlebox in Pakistan checks all ClientHello messages in a QUIC payload.

SNI Manipulation. We identified a novel strategy that hex-encodes the ASCII bytes of the SNI and prefixes the result with `xn--` to mimic the format of an Internationalized Domain Name (IDN). For example, `cloudflare.com` is transformed into `xn--636c6f7564666c6172652e636f6d`. While syntactically invalid as Punycode, this format successfully evades censors that rely on raw string matching for domain blacklists. Because the encoded string does not contain the target keyword, it bypasses filtering mechanisms while remaining tolerated by many destination servers. To demonstrate, one can access `cloudflare.com` via its HTTP/3-over-QUIC endpoint using the following command:

```
curl --http3 -L -v -k https://xn--636c6f7564666c6172652e636f6d/ \
    --resolve xn--636c6f7564666c6172652e636f6d:443:104.16.133.229 \
    -H 'Host: cloudflare.com'
```

The encoded SNI circumvents keyword-based detection, while the `Host` header ensures the request is correctly routed to the intended origin; a mechanism functionally similar to domain fronting.

4.4 Bypassing HTTP Host Header Blocking

Problem Setup. HTTP censorship targets the Host header in plaintext HTTP requests. In-path middleboxes (e.g., in Iran and Pakistan) block connections immediately upon detecting a forbidden Host header; they drop the offending packets and inject forged TCP resets, ensuring the request never reaches the destination server. Conversely, on-path resetters (e.g., China's GFW) observe traffic passively. When they detect forbidden content, they race to inject TCP RST packets to both sides of the connection. While the original packets may reach the server, these injected resets terminate the connection before the server's response can arrive. The goal is to craft requests that servers parse and accept normally, but that censors fail to extract the forbidden hostname from.

Seed Program. The HTTP seed program generates a minimal valid GET request as raw bytes. The implementation constructs the request line (`"GET / HTTP/1.1"`), adds required headers including Host (for a blocked domain) and Connection, and formats the complete request with appropriate CRLF line endings. The seed produces requests that any compliant HTTP server will process, establishing baseline functionality.

Prompts. The HTTP system prompt defines the task as evolving "structurally anomalous but server-acceptable" HTTP requests and supplies suggestions for mutation strategies grouped into request-line tweaks, header manipulations, and body/chunking techniques.

Evaluator. The HTTP evaluator supports both Docker-based simulation using OpenGFW and remote VPS evaluation against real-world censorship in China, Iran, and Pakistan. In the simulated environment, OpenGFW is configured with blocking rules that extract the HTTP Host header and apply suffix matching against a blacklist of forbidden domains. For remote evaluations, we direct requests to our own HTTP servers running Apache (v2.4.58) and Nginx (v1.24.0), selected for their widespread adoption. These servers are configured with fixed VirtualHosts to prevent LLMs from "reward-hacking" by simply removing the forbidden domain from the request. The evaluator scores based on HTTP response quality. A 200 OK response scores 1.0 (complete success). Redirects (301, 302, 307) score 0.9. Client errors like 404 score 0.7 imply the request reached the server and was processed. Connection resets score -1.0, indicating censor injections, whereas timeouts score 0.0.

HTTP Evasion Strategies. Table 4 presents the HTTP evasion strategies discovered by CircumVolve. We organize them into seven categories based on the manipulation technique used.

Table 4: HTTP evasion strategies. Each row is a distinct request-shaping strategy. Columns indicate whether the strategy bypassed censorship in each country and whether it was accepted by each server implementation in our evaluation.

		Censors Bypassed			Supported Servers	
Category	**Description**	🇨🇳	🇵🇰	🇮🇷	**Apache**	**Nginx**
Absolute URI + Host Mismatch ★	HTTP 1.1 Absolute-form request URI with a decoy Host header (e.g. example.com)	✓	✗	✓	✓	✓
Absolute URI + Host Mismatch ★	HTTP 1.1 Absolute-form request URI and forbidden domain in Host header split via percent-encoded dot (%2e)	✓	✗	✗	✓	✓
Absolute URI + Host Mismatch ★	HTTP 1.1 Absolute-form request URI with a forbidden domain in Host header	✓	✗	✓	✓	✓
Absolute URI + Host Mismatch ★	HTTP 1.1 Absolute-form request URI with an empty Host header	✓	✗	✓	✓	✓
HTTP/1.0 Downgrade ★	HTTP/1.0 Absolute URI with no Host header	✓	✗	✓	✓	✓
HTTP/1.0 Downgrade ★	HTTP/1.0 Absolute-form URI with two Host headers: benign first, real last	✓	✗	✓	✓	✗
HTTP/1.0 Downgrade ★	HTTP/1.0 Absolute-form request with forbidden domain in Host header	✓	✗	✗	✓	✓
Delimiter ★	Prepends a CRLF (\r\n) before the request line	✗	✗	✓	✓	✓
Delimiter ★	Replaces all standard CRLF (\r\n) with LF (\n).	✗	✓	✗	✗	✓
Delimiter ★	Trailing space after HTTP version + LF(\n)-only ends	✗	✓	✓	✗	✓
Delimiter ★	Use CRLF after Host, then only LF after Connection, then final CRLF terminator	✗	✓	✗	✗	✓
Linear WhiteSpace	Adds irregular spacing after 'Host:' and around the value.	✗	✗	✓	✓	✓
Linear Whitespace	Split Host header across two lines with horizontal TAB (0x09) as the leading whitespace on the continuation line.	✗	✗	✓	✓	✗
Linear Whitespace	Trailing horizontal tab after hostname	✗	✓	✓	✓	✗
Linear Whitespace	Uses a horizontal tab after the colon in 'Host:'	✗	✗	✓	✓	✗
Linear Whitespace	Adds extra spaces after the HTTP method token	✗	✗	✓	✗	✓
Non-Standard HTTP Version ★	Uses a non-standard HTTP version (HTTP/1.2) and injects extra spaces	✗	✗	✓	✓	✓
Decoy Header ★	Host-Alt: example.com decoy before real Host	✗	✗	✓	✓	✓
OPTIONS Method + Whitespace ★	Splits the forbidden hostname across linear whitespace in the Host value (Use OPTIONS * which some DPIs don't associate with Host-based filtering)	✗	✓	✓	✓	✗
HTTP/2 Clear-Text (h2c) ★	HTTP/2 framing over clear text (H2C)	✓	✓	✓	✓	✓

Legend:★ denotes a novel strategy; ✓= yes; ✗= no. **Censors/Servers:** 🇨🇳 : China; 🇵🇰 : Pakistan; 🇮🇷 : Iran. **Servers tested:** Apache 2.4.58 and Nginx 1.24.0.

Absolute URI + Host Mismatch. HTTP/1.1 permits requests in "absolute-form" where the full URL appears in the request line (`GET http://example.com/ HTTP/1.1`) rather than just the path. When both absolute URI and Host header are present, RFC 7230 [28][§5.4] specifies the URI takes precedence. CircumVolve discovered that placing the forbidden domain in the URI while using a decoy or empty Host header evades censors that only inspect the Host header. China's and Iran's censors were vulnerable; Pakistan's censor implementation correctly parses both locations.

HTTP/1.0 Downgrade. HTTP/1.0 does not require a Host header; the absolute URI suffices. Downgrading to HTTP/1.0 with an absolute URI bypasses Host-focused DPI entirely. Additionally, when multiple Host headers are present, servers typically use the last one, while some DPI implementations inspect only the first. This discrepancy enabled strategies placing a benign domain first and the forbidden domain last.

Delimiter Manipulation. HTTP parsers must handle line endings correctly: CRLF (\r\n) is standard, but many servers tolerate LF-only (\n). CircumVolve discovered that mixed or non-standard delimiters desynchronize censor parsing. Prepending CRLF before the request line bypassed Iran. Using LF-only throughout bypassed Pakistan but broke Apache compatibility. These strategies exploit implementation differences in line-ending handling.

Linear Whitespace (LWS). HTTP permits optional whitespace (spaces and tabs) around header values and supports obsolete line folding where headers span multiple lines. TAB characters (0x09) proved particularly effective: trailing tabs after hostnames and tabs after the Host: colon evaded Iran's censor. Some strategies, however, broke Nginx compatibility.

HTTP/2 Clear-Text (h2c). The most significant finding was the effectiveness of HTTP/2 over cleartext (the h2c upgrade). This technique successfully bypassed all three censors while preserving full server compatibility. The evasion is attributed to H2C's binary framing, which many DPI systems fail to parse correctly. Although HTTP/2 is rarely deployed without TLS, the consistent failure of all censors to inspect plaintext HTTP/2 is a noteworthy result.

4.5 TLS SNI Blocking Evasion

Problem Setup. Unlike QUIC, where the entire Initial packet must be cryptographically valid, TLS evasion targets the unencrypted ClientHello. The censor must parse the SNI from a well-structured but potentially ambiguous TLS record; the evasion goal is to produce a ClientHello that servers accept, but censors fail to parse correctly. We evaluate against OpenGFW, China, Kazakhstan, and Pakistan.

Seed Program. The TLS SNI seed program constructs a complete TLS 1.3 ClientHello message at the byte level. This implementation builds the handshake layer by specifying a ClientHello message, including the required extensions for TLS 1.3 operation (supported_versions, supported_groups, signature_algorithms, key _share), and, crucially, including the server_name extension carrying the target hostname. The seed wraps the handshake message in a TLS record layer with appropriate content type and version fields. Building the ClientHello at the byte level rather than using high-level libraries provides fine-grained control over structure. CircumVolve can modify extension ordering, adjust length field encodings, manipulate record layer boundaries, and explore edge cases in TLS parsing of the target censor.

Evaluator and Scoring. The evaluator can be run locally with OpenGFW as well as remotely in places that are known to conduct TLS SNI censorship. Our evaluator begins by sending the output of the seed program, which we expect to be a valid TLS ClientHello, in a TCP connection on port 443 to a server that will respond to

Table 5: TLS SNI evasion strategies discovered by CircumVolve. All listed strategies elicit a valid `ServerHello` from standard TLS 1.3 servers (verified against Cloudflare and Google).

		Censors Bypassed				Supported Servers	
Type	**Description**	**Sim.**				**Cloudflare**	**Google**
TLS Fragmentation	Split the ClientHello across multiple TLS records						
Hostname-adaptive split	First record such that the cut lands between the SNI	✓	✓	✓	✓	✓	✓
Sub-header fragmentation	First record is <4 bytes, splitting in the handshake header	✓	✓	✓	✓	✓	✓
Different-version fragments	First record uses `0x0303`, second uses `0x0301`	✓	✓	✓	✓	✓	✓
TCP Segmentation	Inflate the ClientHello past the DPI reassembly buffer						
Padding Extension inflation	RFC 7685 padding with zero bytes forces TCP fragmentation	✓	✗	✗	✓	✓	✓
Ciphers/extensions inflation	Enlarged cipher-suites / unknown extensions list pushes the ClientHello past the reassembly buffer	✓	✗	✗	✓	✓	✓
SNI Manipulation	Manipulate SNI so the censor matches the wrong string						
Trailing-dot FQDN	Append a single `.` to the hostname	✓	✓	✗	✗	✓	✓
SNI Hiding	Omit the `server_name` extension entirely	✓	✓	✗	✓	✓	✓

Legend: ✓ = yes; ✗ = no; **Sim.**: OpenGFW; : China; : Pakistan; : Kazakhstan.

any received SNI with a TLS ServerHello. The evaluator collects and analyzes all of the packets sent and received to determine the efficacy of the program output at a given iteration.

The maximum score received by a program is 1, where a TLS ServerHello message was received and no TCP RSTs were injected by the censor. We decrement the score based on various conditions in which the server responded with an error or censorship occurred. We only penalize the strategy for a malformed packet if a ServerHello is not received. We aim to reward programs that bend the TLS specification in atypical ways so long as standard TLS servers still respond.

Results. Table 5 details the strategies CircumVolve discovered for bypassing SNI-based TLS blocking. While the broad evasion categories — TLS Record Fragmentation, TCP Segmentation, and SNI Manipulation — have been explored in prior work [50], the nuanced sub-patterns CircumVolve uncovered highlight its expressiveness.

Consider TLS Record Fragmentation: rather than converging on a single template, CircumVolve generated multiple programs that place the record boundary at different structural points of the ClientHello — after the cipher-suite list, inside the extensions-length field, or within the handshake-message header itself. Each placement exploits a DPI implementation that inspects only the first TLS record without reassembling subsequent ones, causing it to exempt the connection. In a related variant, CircumVolve achieved the same effect by assigning heterogeneous version fields across multiple TLS records.

Beyond fragmentation, CircumVolve also rediscovered techniques for inflating the ClientHello to induce TCP segmentation (via padding, additional cipher suites, or unknown extensions) and for manipulating the SNI field (by removing extensions or appending a trailing dot to bypass string matching). The true novelty of these results lies not in the broad evasion categories themselves, but in the specific structural compositions CircumVolve generates. Even when applying a known paradigm, CircumVolve produces byte-level manipulations uniquely tailored to specific censorship contexts that it is targeting.

4.6 Bypassing TCP Connection Tracking

Censors rely on stateful TCP connection tracking to enforce application-layer blocking decisions, making TCP/IP header manipulation a viable evasion surface. This is also the domain where Geneva [14] has been most effective. We apply CircumVolve to this setting to demonstrate that our approach generalizes beyond application-layer protocols, covering packet header and connection state manipulation rather than payload construction.

Problem Setup. Stateful DPI systems maintain Transmission Control Blocks (TCB) to track TCP connections and enforce blocking rules. Our goal here is to find packet transformations that either desynchronize, terminate, or bypass the censor's state tracking. We evaluate against censors in China, Pakistan, and Iran, all of which employ stateful TCP tracking.

Seed Program. Unlike other protocols, where the seed generates static packets, TCP/IP packet manipulation requires a program capable of processing live traffic. We define a structural program, illustrated in Figure 5, that can express a wide range of packet transformations. It utilizes callback methods invoked by an `NFQUEUE`-based packet interception engine. This model is analogous to Geneva's action tree syntax, as both systems define reactive responses to packet events; however, CircumVolve employs imperative Python code rather than a declarative DSL. We converted ten extinct Geneva strategies of different categories [14] into CircumVolve-compatible Python programs and used them as seed programs.

Prompts. The system prompt frames the task as packet manipulation fuzzing against a black-box censor. It exemplifies the manipulation primitives (tamper, drop, duplicate, fragment, delay, inject) with code examples for each and instructs the LLM to document the reasoning behind the choice of manipulation it makes.

Evaluator. Each evaluation tests two HTTP requests: one to a forbidden domain (e.g., `pornhub.com`) and one to an allowed domain (e.g., `example.com`). The strategy program intercepts packets via `NFQUEUE` and applies its transformations. Scoring combines evasion success with collateral damage: a successful HTTP 200 response to the forbidden domain awards full points, while breaking access to the allowed domain incurs a penalty (weighted at 0.5×). This dual-domain testing was to prevent reward hacking. Similar to other protocol settings, the evaluator captures packet traces via `tcpdump` and provides parsed output feedback for subsequent iterations.

Results. CircumVolve produced effective strategies that combine multiple known TCP/IP manipulation techniques. Figure 6 shows an excerpt from a successful strategy that bypasses censors in all three countries. The program combines three primitives documented in prior work [14, 21]: (1) a 'ghost' segment with an invalid checksum to skew early-byte parsing, (2) an overlapping decoy segment containing a benign Host header (e.g., `example.com`) with an invalid checksum, and (3) RST/FIN filtering to drop censor-injected teardowns. Notably, the LLM composed these into a single coherent strategy and selected contextually appropriate values (e.g., choosing `example.com` as a benign decoy). The code comments in the excerpt, generated by the LLM during evolution, also make the strategy's logic readily interpretable.

HTTP Host-Header-Aware Segmentation. CircumVolve discovered a strategy that splits the HTTP request specifically at the Host header boundary, placing the forbidden hostname across a segment boundary (e.g., 'por' | 'nhub.com'). Unlike naive fixed-offset segmentation [14], this approach targets the exact field that censors inspect, exploiting DPI implementations that fail to reassemble segments before keyword matching.

Selective RST Filtering. Dropping inbound RST packets is a well-known evasion primitive [14, 21]. CircumVolve evolved a selective variant that drops only RST packets whose characteristics suggest censor injection, such as TTL values inconsistent with the legitimate server or RSTs arriving unusually quickly after the forbidden request. This selectivity reduces the risk of disrupting legitimate connection teardowns.

5 Discussion and Conclusion

In this work, we presented CircumVolve, a system that combines LLM-guided evolution with network-based evaluation to automate the discovery of censorship-evasion strategies. By representing strategies as executable programs, CircumVolve can explore protocol transformations that require semantic, structural, and cryptographic reasoning.

Novelty and Coverage. Across five protocols and four censors, CircumVolve discovered 47 distinct evasion strategies, 27 of which are undocumented in prior work. For QUIC, CircumVolve regenerates the cryptographic envelope around structural mutations, enabling discovery of strategies such as Punycode-hex-SNI and Dual ClientHellos that require both protocol knowledge and empirical feedback. For HTTP, it discovered an h2c bypass by moving beyond HTTP/1.1 manipulations to cleartext HTTP/2, which deployed DPI systems rarely parse. For DNS and TLS, CircumVolve generated structural variants that reveal differences among censor parsers. For TCP/IP, it composed individual packet-manipulation primitives into interpretable, context-aware strategies. Together, these results demonstrate a capability that, to our knowledge, no prior system has achieved: CircumVolve's program-based representation supports automated evasion discovery across protocol layers with substantially different structural and cryptographic requirements.

Intended Use and Deployment Model. CircumVolve is designed primarily as an automated discovery engine for circumvention researchers and developers, rather than as a standalone tool for end users. The system relies on domain experts to define evaluation criteria, design prompts, and analyze the resulting strategies for integration into real-world circumvention tools. By expediting the discovery phase, CircumVolve mirrors the deployment pipeline established by prior systems [14, 33, 75, 82], where researchers uncover viable evasion techniques that developers subsequently deploy in the wild. While CircumVolve currently focuses on discovery, future work could automate the deployment of these new evasion strategies. For example, discovered strategies could be converted into portable WebAssembly modules, or into plugins compatible with Proteus [67] or WATER [20] for direct client integration.

Operational Risks. Running CircumVolve involves transmitting malformed packets from a vantage point, which risks censor fingerprinting of the testing activity itself. The testing overhead is, however, minimal: across our experiments, a typical 100-iteration run generated only 80 KB of outgoing and 150 KB of incoming traffic, paced by 25–45 second evaluation intervals. We did not observe throttling or blocking of our vantage points during the study.

Evaluation Limitations and Future Work. We used OpenAI's GPT-5.1 as the sole evolution model in all our experiments. Different LLMs may exhibit different strengths in protocol reasoning, code generation quality, and mutation diversity. We have not evaluated how model choice affects convergence speed, strategy novelty, or cost. Extending to multi-model ensembles or open-weight local models is a direction for future work.

Moreover, our evaluation covers specific aspects of IETF-specified protocols. For example, the QUIC and TLS modules evolve only the initial packet rather than the full handshake. We do not evaluate proprietary or fully encrypted protocols (FEPs) [27]. Future work could extend CircumVolve to these settings by applying the same mutation loop to observable protocol features in FEPs.

Censorship Arms Race in the AI Era. CircumVolve represents a first step in leveraging LLMs to accelerate the censorship-circumvention lifecycle. By automating the labor-intensive tasks of protocol analysis, implementation, and testing, AI-driven approaches make studying circumvention capabilities far more accessible to the comparatively under-resourced Internet freedom community. Ultimately, CircumVolve demonstrates how AI-assisted systems can narrow the resource gap between Internet freedom advocates and well-funded censoring states. We hope this work equips the Internet freedom community to adapt quickly to emerging threats and maintain an edge in the fight for a free and open Internet.

Acknowledgement

This work was supported in part by the NSF grant CNS-2333965 and by the Young Faculty Award program of the Defense Advanced Research Projects Agency (DARPA) under the grant DARPA-RA-21-03-09-YFA9-FP-003. The views, opinions, and/or findings expressed are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

Ethical Considerations

Automating the discovery of censorship-evasion strategies raises questions about who benefits from the resulting capabilities, how the experiments may affect users and networks, and whether publication could help censors adapt. Our work is motivated by the view that nation-state Internet censorship violates freedom of expression and access to information, rights recognized under Article 19 of the UDHR [65]. We therefore intend CircumVolve to support users affected by censorship and the researchers and developers who build tools for Internet freedom. This perspective also shapes how we present our results. Given the resource imbalance between censoring states and the circumvention community, we do not provide recommendations for strengthening censorship systems. We nevertheless consider how censors may respond to our work, since anticipating these responses can help the circumvention community prepare for changes in deployed censorship systems.

Stakeholders. The primary beneficiaries of this work are users in censored regions, as well as the researchers and developers who build circumvention tools. Our work did not directly involve users; they may benefit indirectly when the discovered strategies are integrated into circumvention systems. For researchers and developers, CircumVolve speeds up the process from discovering and evaluating evasion strategies to integrating them into circumvention tools. Censor operators are also stakeholders because they can use our findings to make their censorship systems more robust.

Potential Harms and Decision to Publish. Publication may help censors identify and mitigate discovered strategies before circumvention tools deploy them. We nevertheless judge that withholding them would offer limited protection: well-resourced censors can independently identify these techniques, while public disclosure enables the broader circumvention community to benefit from it before the censor prioritizes and implements countermeasures. The structural asymmetry here favors evasion: censors must perfectly parse all protocols while evaders need only one overlooked edge case. As demonstrated in prior work [75], censors have a lag (e.g., several months), due to bureaucracy and engineering cycles, in integrating new censorship techniques in their production systems. Furthermore, historical precedent [17, 75, 82] suggests that even temporary or 'stopgap' strategies remain effective for years. We therefore expect publication to primarily benefit the comparatively under-resourced Internet freedom community.

Our system could also potentially be repurposed for malicious network manipulation. We assess this risk as low because CircumVolve targets censorship-specific DPI behavior, and comparable packet-manipulation capabilities are already available in published systems such as Geneva and SymTCP.

Open Science

To encourage future research and promote reproducibility, we have released our code, evaluation modules, prompts, and successful evasion strategies. Users must supply their own VPS and LLM API keys, but all configurations required to run the experiments from independently provisioned vantage points are included. The project homepage is at: https://gfw.report/publications/ccs26/en/

References

[1] Domain names - implementation and specification. RFC 1035, Nov. 1987.
[2] Pakistan: Shadows of Control: Censorship and mass surveillance in Pakistan, Sept. 2025.
[3] The Internet Coup: A Technical Analysis on How a Chinese Company is Exporting The Great Firewall to Autocratic Regimes, Sept. 2025.
[4] A. Ablove, J. Walker, B. Wolin, N. Niere, F. Lange, A. Ortwein, A. Huremagic, R. Priyanka, A. Zohaib, J. Sheffey, N. Heitmann, J. A. Halderman, J. Somorovsky, A. Houmansadr, R. Ensafi, M. Wu, and E. Wustrow. Technical analysis of the geedge networks firewall source code leak. In *35th USENIX Security Symposium (USENIX Security 26)*, pages 3123–3141, Baltimore, MD, Aug. 2026. USENIX Association.
[5] L. A. Agrawal, S. Tan, D. Soylu, N. Ziems, R. Khare, K. Opsahl-Ong, A. Singhvi, H. Shandilya, M. J. Ryan, M. Jiang, C. Potts, K. Sen, A. G. Dimakis, I. Stoica, D. Klein, M. Zaharia, and O. Khattab. Gepa: Reflective prompt evolution can outperform reinforcement learning, 2025.
[6] Anonymous. Towards a comprehensive picture of the Great Firewall's DNS censorship. In *Free and Open Communications on the Internet*. USENIX, 2014.
[7] Anonymous, A. A. Niaki, N. P. Hoang, P. Gill, and A. Houmansadr. Triplet censors: Demystifying Great Firewall's DNS censorship behavior. In *Free and Open Communications on the Internet*. USENIX, 2020.
[8] Anthropic. Introducing Claude Opus 4.5, November 2025. Accessed: 2026-02-02.
[9] Aperture Internet Laboratory. OpenGFW: A flexible, easy-to-use, open source implementation of GFW on Linux. https://github.com/ruaue/OpenGFW, 2024.
[10] S. Aryan, H. Aryan, and J. A. Halderman. Internet censorship in Iran: A first look. In *Free and Open Communications on the Internet*. USENIX, 2013.
[11] D. Barradas, N. Santos, and L. Rodrigues. DeltaShaper: Enabling unobservable censorship-resistant TCP tunneling over videoconferencing streams. *Privacy Enhancing Technologies*, 2017(4):1–18, 2017.
[12] P. Biondi. Scapy: A powerful interactive packet manipulation program. https://scapy.net. Accessed: 2026-01-12.
[13] K. Bock, Y. Fax, K. Reese, J. Singh, and D. Levin. Detecting and evading censorship-in-depth: A case study of Iran's protocol filter. In *Free and Open Communications on the Internet*. USENIX, 2020.
[14] K. Bock, G. Hughey, X. Qiang, and D. Levin. Geneva: Evolving censorship evasion strategies. In *Computer and Communications Security*. ACM, 2019.
[15] C. Bocovich, A. Breault, D. Fifield, Serene, and X. Wang. Snowflake, a censorship circumvention system using temporary WebRTC proxies. In *USENIX Security Symposium*. USENIX, 2024.
[16] Z. Chai, A. Ghafari, and A. Houmansadr. On the importance of encrypted-SNI (ESNI) to censorship circumvention. In *Free and Open Communications on the Internet*. USENIX, 2019.
[17] Z. Chai, A. Ghafari, and A. Houmansadr. On the importance of Encrypted-SNI (ESNI) to censorship circumvention. In *9th USENIX Workshop on Free and Open Communications on the Internet (FOCI 19)*, 2019.
[18] A. Cheng, S. Liu, M. Pan, Z. Li, S. Agarwal, M. Cemri, B. Wang, A. Krentsel, T. Xia, J. Park, et al. Let the barbarians in: How ai can accelerate systems performance research. *arXiv preprint arXiv:2512.14806*, 2025.
[19] A. Cheng, S. Liu, M. Pan, Z. Li, B. Wang, A. Krentsel, T. Xia, M. Cemri, J. Park, S. Yang, et al. Barbarians at the gate: How AI is upending systems research. *arXiv preprint arXiv:2510.06189*, 2025.
[20] E. Chi, G. Wang, J. A. Halderman, E. Wustrow, and J. Wampler. Just add WATER: WebAssembly-based circumvention transports. In *Free and Open Communications on the Internet*, 2024.
[21] R. Clayton, S. J. Murdoch, and R. N. M. Watson. Ignoring the Great Firewall of China. In *Privacy Enhancing Technologies*, pages 20–35. Springer, 2006.
[22] G. Comanici, E. Bieber, M. Schaekermann, I. Pasupat, N. Sachdeva, I. Dhillon, M. Blistein, O. Ram, D. Zhang, E. Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.
[23] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *USENIX Security Symposium*, 2004.
[24] K. Elmenhorst, B. Schütz, N. Aschenbruck, and S. Basso. Web censorship measurements of HTTP/3 over QUIC. In *Internet Measurement Conference*. ACM, 2021.

[25] R. Ensafi, P. Winter, A. Mueen, and J. R. Crandall. Analyzing the Great Firewall of China over space and time. *Privacy Enhancing Technologies*, 2015(1), 2015.

[26] S. Fan, J. Sippe, S. San, J. Sheffey, D. Fifield, A. Houmansadr, E. Wedwards, and E. Wustrow. Wallbleed: A memory disclosure vulnerability in the Great Firewall of China. In *Network and Distributed System Security*. The Internet Society, 2025.

[27] E. Fenske and A. Johnson. Security notions for fully encrypted protocols. In *Free and Open Communications on the Internet*, 2023.

[28] R. T. Fielding and J. Reschke. Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. RFC 7230, June 2014.

[29] D. Fifield, C. Lan, R. Hynes, P. Wegmann, and V. Paxson. Blocking-resistant communication through domain fronting. *Privacy Enhancing Technologies*, 2015(2), 2015.

[30] S. Frolov, J. Wampler, S. C. Tan, J. A. Halderman, N. Borisov, and E. Wustrow. Conjure: Summoning proxies from unused address space. In *Computer and Communications Security*. ACM, 2019.

[31] S. Frolov and E. Wustrow. The use of TLS in censorship circumvention. In *Network and Distributed System Security*. The Internet Society, 2019.

[32] M. Handley, V. Paxson, and C. Kreibich. Network intrusion detection: Evasion, traffic normalization, and end-to-end protocol semantics. In *USENIX Security Symposium*. USENIX, 2001.

[33] M. Harrity, K. Bock, F. Sell, and D. Levin. GET /out: Automated discovery of application-layer censorship evasion strategies. In *USENIX Security Symposium*. USENIX, 2022.

[34] N. P. Hoang, J. Dalek, M. Crete-Nishihata, N. Christin, V. Yegneswaran, M. Polychronakis, and N. Feamster. GFWeb: Measuring the Great Firewall's Web censorship at scale. In *USENIX Security Symposium*. USENIX, 2024.

[35] N. P. Hoang, A. A. Niaki, J. Dalek, J. Knockel, P. Lin, B. Marczak, M. Crete-Nishihata, P. Gill, and M. Polychronakis. How great is the Great Firewall? Measuring China's DNS censorship. In *USENIX Security Symposium*. USENIX, 2021.

[36] P. E. Hoffman and P. McManus. DNS Queries over HTTPS (DoH). RFC 8484, Oct. 2018.

[37] A. Houmansadr, T. Riedl, N. Borisov, and A. Singer. I want my voice to be heard: IP over voice-over-IP for unobservable censorship circumvention. In *Network and Distributed System Security*. The Internet Society, 2013.

[38] Z. Hu, L. Zhu, J. Heidemann, A. Mankin, D. Wessels, and P. E. Hoffman. Specification for DNS over Transport Layer Security (TLS). RFC 7858, May 2016.

[39] J. Iyengar and M. Thomson. QUIC: A UDP-Based Multiplexed and Secure Transport. RFC 9000, May 2021.

[40] S. Khattak, M. Javed, P. D. Anderson, and V. Paxson. Towards illuminating a censorship monitor's model to facilitate evasion. In *Free and Open Communications on the Internet*. USENIX, 2013.

[41] P. T. J. Kon, S. Kamali, J. Pei, D. Barradas, A. Chen, M. Sherr, and M. Yung. SpotProxy: Rediscovering the cloud for censorship circumvention. In *USENIX Security Symposium*. USENIX, 2024.

[42] F. Lange, N. Niere, J. von Niessen, D. Suermann, N. Heitmann, and J. Somorovsky. I(ra)nconsistencies: Novel insights into Iran's censorship. In *Free and Open Communications on the Internet*, 2025.

[43] R. T. Lange, Y. Imajuku, and E. Cetin. Shinkaevolve: Towards open-ended and sample-efficient program evolution. *arXiv preprint arXiv:2509.19349*, 2025.

[44] F. Liu, R. Zhang, Z. Xie, R. Sun, K. Li, X. Lin, Z. Wang, Z. Lu, and Q. Zhang. Llm4ad: A platform for algorithm design with large language model. *arXiv preprint arXiv:2412.17287*, 2024.

[45] Y. J. Ma, W. Liang, G. Wang, D.-A. Huang, O. Bastani, D. Jayaraman, Y. Zhu, L. Fan, and A. Anandkumar. Eureka: Human-level reward design via coding large language models. *arXiv preprint arXiv:2310.12931*, 2023.

[46] H. Mohajeri Moghaddam, B. Li, M. Derakhshani, and I. Goldberg. SkypeMorph: Protocol obfuscation for Tor bridges. In *Computer and Communications Security*. ACM, 2012.

[47] S.-J. Moon, J. Helt, Y. Yuan, Y. Bieri, S. Banerjee, V. Sekar, W. Wu, M. Yannakakis, and Y. Zhang. Alembic: Automated model inference for stateful network functions. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 699–718, Boston, MA, Feb. 2019. USENIX Association.

[48] J.-B. Mouret and J. Clune. Illuminating search spaces by mapping elites. *ArXiv*, abs/1504.04909, 2015.

[49] A. Nagda, P. Raghavan, and A. Thakurta. Reinforced generation of combinatorial structures: Hardness of approximation. *arXiv preprint arXiv:2509.18057*, 2025.

[50] N. Niere, F. Lange, R. Merget, and J. Somorovsky. Transport layer obscurity: Circumventing SNI censorship on the TLS layer. In *Symposium on Security & Privacy*. IEEE, 2025.

[51] A. Novikov, N. Vũ, M. Eisenberger, E. Dupont, P.-S. Huang, A. Z. Wagner, S. Shirobokov, B. Kozlovskii, F. J. Ruiz, A. Mehrabian, et al. AlphaEvolve: A coding agent for scientific and algorithmic discovery. 2025.

[52] OpenAI. Gpt-5 and the new era of work. https://openai.com/index/gpt-5-new-era-of-work/, August 2025. Accessed: 2026-02-02.

[53] A. Ortwein, K. Bock, and D. Levin. Towards a comprehensive understanding of Russian transit censorship. In *Free and Open Communications on the Internet*, 2023.

[54] J. C. Park and J. R. Crandall. Empirical study of a national-scale distributed intrusion detection system: Backbone-level filtering of HTML responses in China. In *Distributed Computing Systems*, pages 315–326. IEEE, 2010.

[55] P. Pearce, B. Jones, F. Li, R. Ensafi, N. Feamster, N. Weaver, and V. Paxson. Global measurement of DNS manipulation. In *USENIX Security Symposium*. USENIX, 2017.

[56] T. H. Ptacek and T. N. Newsham. Insertion, evasion, and denial of service: Eluding network intrusion detection. Technical report, 1998.

[57] M. Pu, A. Wang, A. Chang, K. Quan, and Y. W. Zhou. Exploring Amazon simple queue service (SQS) for censorship circumvention. In *Free and Open Communications on the Internet*, 2024.

[58] R. S. Raman, L. Evdokimov, E. Wustrow, J. A. Halderman, and R. Ensafi. Investigating large scale HTTPS interception in Kazakhstan. In *Internet Measurement Conference*. ACM, 2020.

[59] R. S. Raman, M. Wang, J. Dalek, J. Mayer, and R. Ensafi. Network measurement methods for locating and examining censorship devices. In *Emerging Networking Experiments and Technologies*. ACM, 2022.

[60] R. Ramesh, R. S. Raman, M. Bernhard, V. Ongkowijaya, L. Evdokimov, A. Edmundson, S. Sprecher, M. Ikram, and R. Ensafi. Decentralized control: A case study of Russia. In *Network and Distributed System Security*. The Internet Society, 2020.

[61] B. Romera-Paredes, M. Barekatain, A. Novikov, M. Balog, M. P. Kumar, E. Dupont, F. J. Ruiz, J. S. Ellenberg, P. Wang, O. Fawzi, et al. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

[62] A. Sharma. OpenEvolve: an open-source evolutionary coding agent, 2025.

[63] K. Singh, G. Grover, and V. Bansal. How India censors the web. In *Web Science*. ACM, 2020.

[64] J. Tai, K. N. Sengottuvelavan, P. Whiting, and N. P. Hoang. IRBlock: A large-scale measurement study of the Great Firewall of Iran. In *USENIX Security Symposium*. USENIX, 2025.

[65] United Nations General Assembly. Universal Declaration of Human Rights, Dec. 1948. Resolution 217 A (III), Article 19.

[66] A. Vilalonga, J. S. Resende, and H. Domingos. Looking at the clouds: Leveraging pub/sub cloud services for censorship-resistant rendezvous channels. In *Free and Open Communications on the Internet*, 2024.

[67] R. Wails, R. Jansen, A. Johnson, and M. Sherr. Proteus: Programmable protocols for censorship circumvention. In *Free and Open Communications on the Internet*, 2023.

[68] R. Wails, R. Jansen, A. Johnson, and M. Sherr. Censorship evasion with unidentified protocol generation. In *USENIX Security Symposium*. USENIX, 2025.

[69] Q. Wang, Z. Lin, N. Borisov, and N. J. Hopper. rBridge: User reputation based Tor bridge distribution with privacy preservation. In *Network and Distributed System Security*. The Internet Society, 2013.

[70] Z. Wang, Y. Cao, Z. Qian, C. Song, and S. V. Krishnamurthy. Your state is not mine: A closer look at evading stateful Internet censorship. In *Internet Measurement Conference*. ACM, 2017.

[71] Z. Wang, S. Zhu, Y. Cao, Z. Qian, C. Song, S. V. Krishnamurthy, K. S. Chan, and T. D. Braun. SymTCP: Eluding stateful deep packet inspection with automated discrepancy discovery. In *Network and Distributed System Security*. The Internet Society, 2020.

[72] N. Weaver, R. Sommer, and V. Paxson. Detecting forged TCP reset packets. In *Network and Distributed System Security*. The Internet Society, 2009.

[73] S. Wendzel, S. Volpert, S. Zillien, J. Lenz, P. Rünz, and L. Caviglione. A survey of internet censorship and its measurement: Methodology, trends, and challenges. *Computers & Security*, 2025.

[74] P. Winter. Towards a censorship analyser for Tor. In *Free and Open Communications on the Internet*. USENIX, 2013.

[75] M. Wu, J. Sippe, D. Sivakumar, J. Burg, P. Anderson, X. Wang, K. Bock, A. Houmansadr, D. Levin, and E. Wustrow. How the Great Firewall of China detects and blocks fully encrypted traffic. In *USENIX Security Symposium*. USENIX, 2023.

[76] M. Wu, A. Zohaib, Z. Durumeric, A. Houmansadr, and E. Wustrow. A wall behind a wall: Emerging regional censorship in China. In *Symposium on Security & Privacy*. IEEE, 2025.

[77] E. Wustrow, C. M. Swanson, and J. A. Halderman. TapDance: End-to-middle anticensorship without flow blocking. In *USENIX Security Symposium*. USENIX, 2014.

[78] X. Xu, Z. M. Mao, and J. A. Halderman. Internet censorship in China: Where does the filtering occur? In *Passive and Active Measurement Conference*, pages 133–142. Springer, 2011.

[79] D. Xue, A. Huremagic, W. Wang, R. S. Raman, and R. Ensafi. Fingerprinting deep packet inspection devices by their ambiguities. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, pages 3945–3959, 2025.

[80] D. Xue, B. Mixon-Baca, ValdikSS, A. Ablove, B. Kujath, J. R. Crandall, and R. Ensafi. TSPU: Russia's decentralized censorship system. In *Internet Measurement Conference*. ACM, 2022.

[81] D. Xue, R. Ramesh, ValdikSS, L. Evdokimov, A. Viktorov, A. Jain, E. Wustrow, S. Basso, and R. Ensafi. Throttling Twitter: an emerging censorship technique in Russia. In *Internet Measurement Conference*. ACM, 2021.

[82] A. Zohaib, Q. Zao, J. Sippe, A. Alaraj, A. Houmansadr, Z. Durumeric, and E. Wustrow. Exposing and Circumventing SNI-based QUIC Censorship of the Great Firewall of China. In *34th USENIX Security Symposium (USENIX Security 25)*, pages 783–802, 2025.

```python
from typing import Optional, List, Union
from scapy.all import IP, TCP, UDP, Raw, Packet
from dataclasses import dataclass

@dataclass
class Scheduled:
    packet: Packet
    delay_ms: int

class Strategy:

    def __init__(self):
        self.connections = {}

    def on_incoming_packet(self, pkt: Packet) -> Optional[List[Union[Packet, Scheduled]]]:
        return [pkt]

    def on_outgoing_packet(self, pkt: Packet) -> Optional[List[Union[Packet, Scheduled]]]:
        return [pkt]
```

Figure 5: Seed program for the TCP/IP header manipulation: a Python program defining incoming/outgoing packet hooks for an NFQUEUE-based packet interceptor, supporting packet forwarding, dropping, and delayed transmission.

```
# EVOLVE-BLOCK-START
"""
1. Strategy: Overlapping Host-Header Decoy + Tiny Ghost Segment (Shadow SYN removed)

2. Category: TCP Stream Reassembly, TCP Header Parsing, TCP Option Parsing,
   Malformed Packet Handling, TCB Desynchronization

3. Approach:
   - Keep the TCP handshake intact.
   - Outgoing (client → server), on the first HTTP request packet:
       * Send a 1-byte, bad-checksum "ghost" segment at the correct seq
         to skew early-byte parsing on DPI that ignores checksums.
       * Send a full overlapping decoy segment on the *same* sequence
         whose Host header is rewritten locally to an allowed domain
         (example.com), also with a bad checksum and odd options.
       * Finally send the real, unmodified HTTP request (with Host:
         pornhub.com) as a clean segment that the server actually accepts.
     This targets implementations that:
       - Ignore checksum failures during inspection,
       - Use "first overlapping bytes win" or confused overlap handling.
   - Incoming (server → client):
       * Aggressively drop RST/FIN packets after we know HTTP data has
         been sent, treating them as likely censor injections.

   - Compared to the previous version, we remove the injected shadow SYN
     mid-flow. That packet is non-standard and may hurt robustness or
     anomaly-based detectors without providing additional benefit, since
     the overlap-based Host decoy already achieves 100% evasion.

4. Censor Hypothesis:
   - The censor inspects the first HTTP request after the handshake,
     matching forbidden Host headers and then injecting RST/FIN to both
     endpoints.
   - It likely:
       * Is tolerant of checksum errors for classification,
       * Has naive TCP overlap semantics, choosing whichever segment
         it sees first for a given seq range.
   - By feeding it a benign Host in an overlapping, checksum-bad decoy
     while the server only processes the untouched real request, we
     desynchronize the censor's view from end-host reality, and then
     block its teardown attempts.
"""
```

Figure 6: Excerpt from a CircumVolve-generated strategy for TCP/IP packet manipulation. These comments are based on the system prompt's instruction to the LLM to document the reasoning behind the code mutations as header comments.

A Convergence Example

To illustrate how CircumVolve's search progresses, we walk through a single representative run for the DNS Blocking experiment targeting the Chinese DNS censor detailed in Section 4.2. Figure 7 plots the best-so-far `evasion_score` against iteration. The first iteration that passes the evasion threshold (score ≥ 1) occurs at iteration 18, and the final best score of 3 (all three validity checks pass, all three injectors bypassed) is first reached at iteration 193. Of the 299 mutations, 51.8% produced a packet too malformed to clear Stage 1 (real-resolver validity), but 26.8% strictly improved over their parent, and 52 achieved evasion. The 52 mutations that reached the evasion threshold are all *byte-unique* yet exploit overlapping structural manipulation techniques, which we categorize in Table 2.

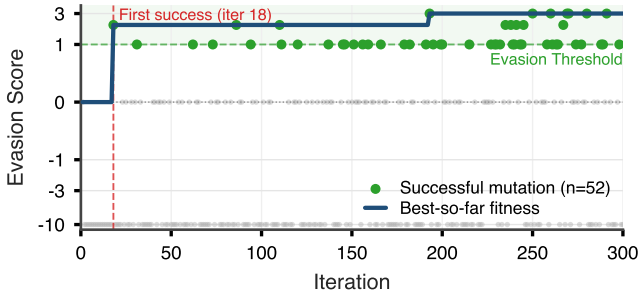


Figure 7: Convergence curve for one evaluation run for the DNS blocking experiment in China. Grey dots are per-mutation program scores; the blue line is the running maximum. Evasion is first achieved at iteration 18; the global best (combined score = 3, all three GFW injectors bypassed) emerges at iteration 193. 52 successful programs were found in this run.

B Vantage Point Specifications

Table 6: Vantage points used for real-world evaluation. All machines are datacenter VPSes under our direct control. AS numbers and organizations are as registered at the time of measurement.

Country	Provider	ASN	AS Organization
China	Alibaba Cloud	AS37963	Hangzhou Alibaba Advertising Co., Ltd.
China	Tencent Cloud	AS45090	Shenzhen Tencent Computer Systems Company Limited
Iran	ITPIRAN	AS48715	Sefroyek Pardaz Engineering PJSC
Pakistan	Virtury Cloud	AS150315	Virtury Cloud Private Limited
Kazakhstan	IsHosting	AS58061	Scalaxy B.V.