International Journal of Science, Strategic Management and Technology
Volume **02** Issue **04** April-**2026** | ISSN: **3108-1762 (Online)** | Impact Factor: 3.8
An International, Peer-Reviewed, Open Access Scholarly Journal Indexed in recognized academic databases

# AI Vs. AI: Generative Adversarial Networks (Gans) for Dynamic Honeypot Generation in Next-Generation Cyber Defense

**Rohit Kumar**
Department of Computer Science
Institute of Information Technology & Management New Delhi, India
*rohit.kumar.it@iitmipu.ac.in*

**Rahul Kumar**
Department of Computer Science
Institute of Information Technology & Management New Delhi, India
*rahul.kumar.it@iitmipu.ac.in*

https://doi.org/10.55041/ijsmt.v2i4.645

**Cite this Article**: Kumar, R. & Kumar, R. (2026). AI Vs. AI: Generative Adversarial Networks (Gans) for Dynamic Honeypot Generation in Next-Generation Cyber Defense. International Journal of Science, Strategic Management and Technology, *02*(04). https://doi.org/10.55041/ijsmt.v2i4.645



***Abstract*—**Honeypots have long served as a practical tool for cyber deception and threat intelligence collection, but their usefulness is bounded by a handful of stubborn problems. They tend to be static, they require careful manual configuration, and modern attackers — especially those equipped with AI-driven reconnaissance — can fingerprint them with surprising ease. This paper studies an alternative: using Generative Adversarial Networks (GANs) to automatically produce honeypot configurations that are dynamic, context-aware, and statistically difficult to distinguish from genuine systems. We refer to this approach as HoneyGAN Pots, and we argue it represents a meaningful shift away from reactive, signature-based deception toward a more proactive, generative form of defense.Our discussion is organized around four threads: an architectural framework that frames decoy generation as an adversarial game; a comparison with traditional static approaches; an honest accounting of the implementation challenges, including computational cost and mode collapse; and a roadmap for integrating future work with reinforcement learning and blockchain-based verification. By treating cyber defense as a contest between an attacking AI and a defending AI, we contend that GAN-generated honeypots provide a scalable, adaptive option that fits the threat surface of Industry 5.0, smart cities, and critical-infrastructure environments.

Index Terms — Generative Adversarial Networks (GANs); Honeypots; Cyber Deception; Adversarial Machine Learning; Dynamic Defense; Network Security; AI vs. AI.

## I. INTRODUCTION

### *A. The Evolving Cyber Threat Landscape*

Digital transformation has expanded the attack surface across nearly every domain of modern life — enterprise networks, personal devices, public infrastructure, and the connected systems that increasingly run our cities. As organizations move toward Industry 5.0, smart-city deployments, and dense Internet-of-Things (IoT) ecosystems, the volume and sophistication of cyber threats have grown in parallel [29].

Conventional perimeter defenses — firewalls, signature-based intrusion detection, antivirus tools — are showing their age. They were not designed for the kinds of attacks now common in the wild: advanced persistent threats (APTs), zero-day exploits, and adversaries who themselves rely on machine learning [11], [29]. Today's attackers automate reconnaissance, evade static detection rules, and continuously mutate their payloads. Defenders, in turn, must protect against an effectively unbounded variety of attacks while operating with finite human and computational resources. The result is an asymmetry that has pushed the field into what many now describe as an AI-versus-AI phase of the cyber arms race.

### *B. The Strategic Purpose of Honeypots*

Honeypots are decoy systems or resources placed in a network to attract and engage malicious actors [2], [7]. They serve four overlapping strategic functions:

- Detection: providing early warning of reconnaissance activity.
- Diversion: consuming an attacker's time and computational effort.
- Intelligence gathering: capturing tools, techniques, and procedures (TTPs).
- Forensics: producing data that informs the design of stronger defenses.

Traditionally, honeypots are split into two broad classes. Low-interaction honeypots emulate a narrow set of services

and are most effective during the reconnaissance phase of the cyber kill chain. High-interaction honeypots are full systems that engage attackers at later stages — delivery, exploitation, lateral movement — and yield richer intelligence at the cost of greater risk and operational overhead.

Since the early 1980s, and especially through the work of efforts such as The Honeynet Project [6], deception-based defense has proven its worth in studying botnets, APT campaigns, and attacker behavior more broadly. Even so, the basic recipe has not changed much in two decades.

### *C. Problem Statement: The Limits of Static Deception*

Despite their value, traditional honeypots carry a number of structural limitations that have only become more pronounced as adversaries grow more capable.

Static configurations. Most honeypots run on hand-built, fixed decoy images. Once deployed they do not change, which means a patient adversary can probe them systematically and build a fingerprint.

Fingerprintability. Sophisticated attackers now use automated tools that look for the tell-tale signs of a decoy: unrealistic file structures, default configurations, atypical network responses, or the absence of normal user activity [13].

Scalability constraints. Maintaining a library of pre-built images and configuration profiles is labor-intensive. It does not extend gracefully to large, heterogeneous environments.

Adaptability gap. As attackers refine their detection techniques, static honeypots cannot adjust on their own. They cannot respond to changes in the surrounding network, the attacker's profile, or current threat intelligence.

These limits add up to a single critical vulnerability. If an attacker can reliably tell a decoy from a real system, the deception strategy has already failed; the attacker simply walks past the honeypot and continues toward production assets.

### *D. Research Objectives and Contributions*

This paper examines a generative alternative. We propose using GANs [1] to automatically produce dynamic, context-aware, and difficult-to-fingerprint honeypot configurations
— a methodology we call HoneyGAN Pots [8]. Conceptually, this represents a move from a static, defender-driven model of deception to one in which the decoy itself is the output of an adaptive learning process.

Our objectives are as follows:

- Establish a conceptual framework for applying GANs to honeypot generation, framing cyber defense as an adversarial game between attacking and defending AI systems.
- Propose a technical architecture for GAN-based decoy generation, including the data model, network structure, and training protocol.
- Compare this approach with traditional methods along the dimensions of adaptation, scalability, and realism.
- Identify implementation challenges and limitations honestly, so the assessment is grounded rather than aspirational.
- Outline directions for future research in GAN-powered cyber defense.

The contributions of this paper are: a synthesis of the emerging literature on GAN-generated honeypots; a detailed architectural sketch for HoneyGAN systems; an articulation of the open research gaps; and practical considerations for both researchers and operators.

The remainder of the paper is organized as follows. Section II reviews relevant background on GANs and honeypots. Section III presents the HoneyGAN architecture and methodology. Section IV analyzes the advantages of the approach. Section V discusses the implementation challenges and limitations. Section VI examines evaluation strategies. Section VII proposes future directions, and Section VIII concludes.

## II. BACKGROUND AND LITERATURE REVIEW

### *A. Generative Adversarial Networks: Architecture and Principles*

Generative Adversarial Networks, introduced by Goodfellow and colleagues in 2014 [1], marked a shift in how researchers thought about generative modeling. Where earlier generative methods focused on maximizing likelihood, GANs train two neural networks against one another in a minimax game.

The Generator (G) takes a random noise vector as input and produces a synthetic sample intended to resemble the true

data distribution. Its goal is to fool the discriminator into accepting its outputs as real.

The Discriminator (D) receives a mix of real samples and generator outputs, and learns to classify each as either authentic or synthetic.

The two networks train simultaneously according to the value function:

$$\min_G \max_D V(D, G) = E_{x \sim p_r(x)}[\log D(x)] + E_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

Here $p_r(x)$ is the real data distribution, $p_z(z)$ is a prior over the noise space, $G(z)$ is the generator's output, and $D(x)$ is the discriminator's estimate that x came from the true distribution. As training progresses, the generator learns to produce more convincing samples while the discriminator becomes a sharper classifier; the two pull each other forward.

Many variants of the original architecture now exist. Conditional GANs (CGANs) attach a label or feature vector to both networks, enabling controlled generation [3]. Wasserstein GANs (WGANs) replace the Jensen–Shannon divergence with the Wasserstein distance and improve training stability [4]. Deep Convolutional GANs (DCGANs) adopt convolutional layers tailored for image generation [5]. For our purposes, the conditional variant matters most: it allows a defender to ask the generator for a decoy of a particular type rather than a generic configuration.

### *B. Honeypots: Classification and Evolution*

Honeypots have been the subject of active research since the late 1980s [2], [7], and they are typically classified along two axes: interaction level and purpose.

| Type | Description | Advantages | Limitations |
|---|---|---|---|
| Low-interaction | Emulates specific services with limited functionality. | Low risk; easy to deploy; modest resource use. | Limited realism; relatively easy to detect. |
| High-interaction | Full systems running real operating systems and applications. | High realism; richer intelligence collection. | Higher risk; resource-intensive; harder to manage. |

By purpose, honeypots fall into research deployments — built primarily to study attacker TTPs, botnets, and APT campaigns — and production deployments, which protect organizational assets by diverting attackers and providing early warning.

It is also useful to think of honeypot technology as having gone through several generations. The first generation, in the late 1980s and 1990s, consisted of simple decoys with manual configuration and limited logging. The second generation, in the 2000s, was shaped heavily by The Honeynet Project [6], which standardized deployment and improved data capture. The third generation, beginning in the 2010s, brought virtualization, automated deployment, and integration with threat intelligence feeds. The fourth and current generation
— still emerging — is characterized by AI-generated, context-aware deception. Across all four generations, one underlying question has remained unresolved: how do we build decoys that are operationally indistinguishable from real production systems while remaining adaptive to changing attacker behavior?

### *C. The Intersection of GANs and Cybersecurity*

GANs have found a home in cybersecurity on both sides of the line. On the offensive side, they have been used to generate adversarial examples that evade machine-learning-based intrusion detection systems [11], [17], to create polymorphic malware variants, to produce convincing deepfakes for social engineering, and to draft realistic phishing content. On the defensive side, GANs help in augmenting imbalanced datasets so that rare attack types can be learned, in adversarial training that hardens models [9], [18], in synthesizing network traffic for safe testing, and in privacy-preserving data sharing.

This dual-use character is a recurring theme in the literature, and it is one of the reasons GANs are now central to the cybersecurity research conversation: the same technology that helps an attacker mutate a payload can also help a defender build deception that the attacker cannot reliably fingerprint.

### *D. Related Work on GANs for Deception and Defense*

A handful of recent studies have begun to explore the direct application of GANs to cyber deception.

HoneyGAN Pots. The 2024 work by Gabrys, Silva, and Bilinski [8] is the foundational study on this topic. They show that a GAN can learn the distribution of network device configurations from real data and produce high-quality

replicas, and they introduce conditional variants that can be steered by operating system or service type. Crucially, the resulting decoys evaded existing honeypot detectors more effectively than static baselines.

MMHP-GAN. A Chinese research team proposed the Mimicry Honeypot GAN [10], which generates features that blend two strategies: protective coloration, in which honeypots mimic real services, and warning coloration, in which real production services are made to resemble honeypots. Their experiments report measurable gains over earlier feature-generation schemes.

Systematic reviews. In 2025, Ndayipfukamiye and colleagues conducted a systematic review of 185 peer-reviewed studies on GAN-based defenses [9], organizing the field along four axes — defensive function, GAN architecture, cybersecurity domain, and adversarial threat model — and noting substantial progress in network intrusion detection, malware analysis, and IoT security.

Adjacent applications. More recent contributions, including GAN-IF for intrusion detection and AR-GAN for autonomous-vehicle defense [28], suggest that GAN-based methods are spreading from controlled research settings into real-time threat mitigation.

### *E. Existing Research Gaps*

Despite this growing interest, several gaps in the literature remain open.

- Limited application to honeypots specifically. Until the work of Gabrys and colleagues [8], no published study had directly applied GANs to the generation of decoy configurations.
- Lack of standardized evaluation. The field does not yet have shared benchmarks or evaluation protocols, which makes results across studies difficult to compare.
- Deployment-readiness questions. Most current work is at the proof-of-concept stage; comparatively little attention has been paid to compute cost, latency, or integration with existing security operations.
- Explainability deficit. GANs are notoriously hard to interpret, which is awkward for security analysts who need to understand why a particular decoy was generated.
- Ethical framework. The dual-use risks of GAN-powered deception are mentioned in passing in many papers but have not been treated rigorously.

Our aim in the remainder of this paper is to consolidate what is currently known, surface these gaps clearly, and sketch a research path that can move GAN-generated honeypots from academic curiosity toward operational practice.

## III. THE HONEYGAN FRAMEWORK: ARCHITECTURE AND METHODOLOGY

### *A. Deception as an Adversarial Game*

The HoneyGAN framework rests on a reframing of cyber deception. Rather than treating defense as a set of static artifacts to be deployed and maintained, we treat it as a game played between two AI systems.

In the traditional setup, the defender manually configures static decoys; the attacker probes the network and uses automated tools to identify them; the attacker then refines their detection methods, while the defender responds with manual updates. This is an asymmetric game, and the asymmetry favors the attacker.

In the HoneyGAN setup, the defender is a generator network that continuously produces new decoy configurations, and the attacker is treated as a discriminator that attempts to separate decoys from real systems. Both adapt continuously, in something approximating real time. The defender's outputs evolve at roughly the same pace as the attacker's detection methods. This is closer to a fair game, and it puts generative capacity at the center of defensive strategy.

### *B. System Architecture*

The HoneyGAN architecture has three primary components: a generator, a discriminator, and the adversarial training loop that connects them.

*Generator Network (the "Forger")*

- Input: a random noise vector, optionally paired with a conditioning input (such as a desired OS type).
- Output: a synthetic network device configuration represented as a feature vector covering ports, services, and OS characteristics.
- Architecture: a deep neural network, typically based on multi-layer perceptrons or convolutional layers depending on how the configuration is represented.

- Objective: produce configurations that the discriminator classifies as real.

The generator's job is to internalize the distribution of legitimate network configurations from the training data. By varying the input noise vector, it can produce a large number of distinct, plausible configurations.

*Discriminator Network (the "Cop")*

- Input: either a real configuration drawn from training data, or a synthetic configuration produced by the generator.
- Output: a probability that the input is real rather than generated.
- Architecture: a deep neural network, typically symmetric in capacity to the generator.
- Objective: classify configurations correctly as real or generated.

The discriminator effectively plays the role of an automated honeypot detector. As it improves, it forces the generator to produce decoys that are statistically harder to separate from genuine systems.

*Adversarial Training Dynamics*

During training, the two networks proceed in lockstep:

1. The generator produces a batch of synthetic configurations from sampled noise.
2. The discriminator evaluates a mixture of real and synthetic configurations.
3. Each network receives gradient feedback derived from the discriminator's classification — the generator on how to make its outputs more convincing, the discriminator on how to be a better classifier.
4. The cycle repeats.

The pressure imposed by this loop is precisely what makes the framework attractive for cyber deception. The generator is forced to keep up with an adversary, which mirrors the dynamic that defenders face in the wild.

### C. Data Model and Feature Representation

For the network device case, Gabrys and colleagues [8] use a relatively simple data model. Each device is described by two pieces of information:

- The set of services exposed on each open port (for example, HTTP on port 80 or SSH on port 22).
- The operating system signature (for example, Windows Server 2019 or Ubuntu 20.04).

These features can be encoded in several ways: categorical variables for OS types and service names, numerical variables for port numbers and version strings, and binary indicators for the presence or absence of specific capabilities or vulnerabilities. Real-world training data can be sourced from active scanning of production systems, from public device-fingerprint repositories, from logs of previously deployed honeypots, and — when needed — from synthetically generated baselines used to bootstrap training.

### D. Conditional GAN Variants for Targeted Deception

Conditional GANs allow the defender to ask for decoys with specific characteristics, which is operationally important: a generic decoy is rarely the right tool for a specific environment.

*Operating-system-conditioned generation*

- Conditioning variable: target OS type (Windows, Linux, macOS, network appliances).
- Use case: deploying decoys that match the OS distribution already present in the network.
- Benefit: the resulting decoys align with the organization's actual technology stack, which improves believability.

*Service-type-conditioned generation*

- Conditioning variable: service category (web servers, databases, file shares, IoT devices).
- Use case: building decoys that attract attackers focused on a particular service class.
- Benefit: enables strategic placement informed by current threat intelligence.

In both cases, the generator receives the noise vector together with the conditioning vector, and is asked to produce configurations that satisfy the constraint while preserving overall realism.

### E. Training Protocol and Hyperparameters

The hyperparameters that follow are taken from the HoneyGAN Pots reference implementation [8].

| Parameter | Value | Description |
|---|---|---|
| Batch size | 64 | Configurations processed per training iteration. |

| Parameter | Value | Description |
|---|---|---|
| Discriminator updates per step | 3 | The discriminator is trained more often than the generator. |
| Gradient penalty coefficient | 10 | Keeps the L2 norm of the discriminator's gradients close to 1. |
| Optimizer | Adam | Adaptive moment estimation. |
| Learning rate | Tuned | Controls the magnitude of parameter updates. |
| $\beta_1$, $\beta_2$ | 0.5, 0.999 | Decay rates used by the Adam optimizer. |

The number of training steps depends on the variant: roughly 11,844 for the unconditional GAN, 5,922 for the OS-conditional GAN, and 23,688 for the device-type-conditional GAN. Training proceeds until the generator reliably fools the discriminator — at which point, by the discriminator's learned criteria, the generated configurations are no longer separable from real ones.

## IV. ADVANTAGES OF GAN-GENERATED HONEYPOTS

The HoneyGAN framework offers several practical advantages over traditional honeypot deployment, and these advantages map fairly cleanly onto the limitations introduced in Section 1.3.

### *A. Dynamic Deception and Moving-Target Defense*

Traditional honeypots, once deployed, do not change. That predictability is what allows attackers to fingerprint them through systematic probing. GAN-generated honeypots are dynamic by construction: the generator can produce new configurations continuously, or on demand, so no two decoys are identical and the same decoy does not persist forever.

This is a natural fit for moving-target defense (MTD) [12], a strategy that aims to increase adversary uncertainty by continuously shifting the attack surface. In the HoneyGAN setting, MTD takes three concrete forms: configuration diversity, in which each decoy presents a unique combination of ports, services, OS fingerprints, and behavioral patterns; temporal dynamics, in which decoys are regenerated on a schedule or in response to detected reconnaissance; and adaptive evolution, in which the discriminator's growing detection ability forces the generator to produce more sophisticated outputs over time.

The practical effect is that attackers cannot rely on pre-computed fingerprints or on commodity detection tools. Each interaction must be analyzed afresh, which raises the cost of reconnaissance considerably.

### *B. Scalability and Automation*

Manual honeypot deployment imposes a clear bottleneck: every decoy needs to be configured, deployed, and maintained by hand if it is to look credible. That bottleneck limits the scale at which deception can operate.

Once trained, a HoneyGAN generator produces an arbitrary number of unique decoy configurations without further human intervention. The contrast with the traditional approach is summarized below.

| Aspect | Traditional honeypots | GAN-generated honeypots |
|---|---|---|
| Configuration effort | Manual, per decoy | Automated; bulk generation |
| Deployment rate | Limited by human resources | Limited primarily by compute |
| Diversity | Constrained by human creativity | Combinatorial; effectively unbounded |
| Maintenance overhead | High (manual updates) | Lower (periodic retraining) |

This kind of scalability enables broader coverage of large enterprise networks, fine-grained placement across subnets, VLANs, and cloud environments, and rapid replacement of compromised or fingerprinted decoys without operational disruption.

### *C. Realism and Unfalsifiability*

Sophisticated attackers detect honeypots by hunting for inconsistencies — unrealistic file timestamps, missing system logs, default configurations, atypical network latency. GANs are designed precisely to avoid these tells: a well-trained generator captures the statistical structure of real systems, and its outputs are, by construction, hard to separate from genuine configurations under the criteria the discriminator has learned.

The realism gain rests on three properties. Statistical fidelity ensures that generated configurations match the distribution of real systems across features such as port combinations, service versions, and OS fingerprints. Contextual

appropriateness, achieved through conditioning, ensures that the generator produces decoys that fit the surrounding environment — Windows configurations in Windows-heavy networks, IoT signatures in smart-building deployments. Emergent realism arises because GANs learn the joint distribution of all features rather than each feature in isolation; they pick up subtle correlations that hand-crafted configurations tend to miss, such as which service combinations co-occur in practice or which OS quirks are characteristic of a given vendor and version.

Together, these properties make GAN-generated honeypots difficult to flag using standard statistical detection methods: the decoys do not deviate from expected patterns because they are themselves drawn from those patterns.

### *D. Adaptability to Evolving Attack Strategies*

Under the traditional model, when attackers learned to detect a particular decoy, defenders had to update their configurations by hand — and during that update window, the decoys were largely useless. GANs change the shape of this problem. Because the generator is constantly being pushed by the discriminator, the system can be retrained as new detection techniques are observed, without rebuilding the deception infrastructure from scratch.

In practice this enables a few useful patterns. Newly discovered detection techniques can be folded into the discriminator's training data, after which the generator is forced to produce decoys that defeat them. Periodic retraining on current production data prevents decoys from drifting out of date as the real environment evolves. And in more advanced deployments, the system can react to active probing by regenerating the decoy under attack, which interferes with the attacker's reconnaissance in a way that static deployments simply cannot.

### *E. Resource Optimization*

High-interaction honeypots are expensive: full operating systems, application stacks, and the monitoring infrastructure to capture attacker behavior. That cost limits how broadly they can be deployed. GAN-generated configurations can be used flexibly across the honeypot spectrum to mitigate this constraint.

Lightweight emulators can be configured directly from generated outputs, reducing the need for full system instances when low-interaction coverage is sufficient. In high-interaction settings, generated configurations can be used to decide where to deploy the more expensive instances — placing them strategically rather than uniformly. Hybrid approaches use generated fingerprints for initial engagement, escalating to a full high-interaction environment only when an attacker's behavior justifies the additional cost. The net result is a more efficient allocation of defensive resources, with broad coverage from cheap decoys and targeted depth where it matters.

## V. IMPLEMENTATION CHALLENGES AND LIMITATIONS

The promise of GAN-generated deception comes with real engineering and methodological costs. We address them here directly so that the assessment is not unbalanced.

### *A. Computational Complexity and Resource Requirements*

GAN training is computationally demanding. It typically requires GPUs or TPUs and nontrivial training time, both of which constrain where and how the technology can be deployed.

| **Resource** | **Typical requirement** | **Practical implication** |
|---|---|---|
| GPU memory | 8–16 GB minimum | Limits batch size and model complexity. |
| Training time | Hours to days per model | Slows the response cycle to new threats. |
| Inference latency | Milliseconds to seconds | May delay on-demand decoy deployment. |
| Energy consumption | High | Operating cost and environmental footprint. |

These costs have deployment consequences. Centralized generation is realistic for enterprise security operations centers with dedicated machine-learning infrastructure. Edge deployment — IoT, edge gateways, resource-constrained devices — is harder and may require quite different model architectures. And true real-time adaptation is bounded by training cycles, which means a millisecond-level response to an active probe is not yet practical with full retraining loops.

Several mitigations are plausible, including pre-trained generators that are fine-tuned rather than trained from scratch, distilled or otherwise lightweight architectures,

cloud-based generation services, and scheduled regeneration rather than continuous adaptation.

### *B. Training Instability and Mode Collapse*

GANs are well known to be hard to train [14]. They are susceptible to instability and to mode collapse, in which the generator settles on a small set of outputs rather than capturing the diversity of the real distribution. In a security context, mode collapse is particularly damaging.

If the generator converges on a single "good enough" configuration that fools the current discriminator, it tends to repeat variations of that configuration. Deployed at scale, this means a fleet of decoys that share an underlying structure, which an attacker can identify after a small number of observations and then bypass everywhere.

Training instability typically surfaces in three ways: non-convergence, in which generator and discriminator never reach a stable equilibrium; vanishing gradients, in which the discriminator becomes too good and stops providing useful learning signal; and oscillation, in which performance fluctuates without trending toward improvement.

There are reasonable mitigations. Wasserstein GANs with a gradient penalty are noticeably more stable than the original formulation [4]. Spectral normalization can constrain the discriminator's capacity so it does not overpower the generator [15]. Experience replay helps preserve diversity across training. Regular evaluation against explicit diversity metrics helps catch mode collapse early, before a deployment is built on top of a degenerate generator.

### *C. Data Scarcity and Quality Requirements*

GANs need a sizeable corpus of high-quality training data to learn realistic distributions. Sourcing that data from production networks raises a few practical issues — privacy, sensitivity of configuration details, and the cold-start problem for organizations that do not have a long history of honeypot deployments to draw on.

Several approaches help. Synthetic data can be used to bootstrap an initial model. Transfer learning from pre-trained models on public datasets can shorten training. Collaborative data sharing, possibly under privacy-preserving protocols such as differential privacy [26] or federated learning [22], allows organizations to pool resources without exposing raw configurations. None of these is a complete solution, but in combination they make the data problem tractable.

### *D. Evaluation Metrics and Validation Difficulties*

A more subtle problem is deciding when a generated honeypot is good enough. The standard GAN evaluation metrics do not transfer well to this domain. Inception Score
[24] was designed for natural images and is not meaningful for network configurations. Fréchet Inception Distance [25] assumes a pre-trained feature extractor of the kind that is widely available for images but not for network data.

Security-specific evaluation needs to capture different properties altogether: detection resistance (does the decoy evade existing honeypot detectors?), engagement rate (do attackers interact with it as they would with a real system?), deception longevity (how long before an attacker identifies it?), and intelligence value (does it produce useful TTP data?).

Validation in practice tends to involve adversarial testing against current honeypot detectors, controlled red-team exercises, A/B comparisons against traditional honeypots, and longer-term observational studies of attacker behavior. None of these is cheap, and the absence of shared benchmarks makes the results difficult to compare across studies — a point we return to in Section VII.

### *E. Adversarial Discovery and GAN Poisoning*

If attackers learn that an organization is using GAN-based deception, they may target the underlying machine-learning system rather than the deployed decoys [11], [18]. Three attack vectors stand out.

GAN poisoning. Attackers seed the training data with corrupted or misleading samples, with the goal of training the generator to produce configurations that are subtly unrealistic or recognizable.

Discriminator evasion. Attackers develop techniques specifically designed to detect GAN-generated configurations by exploiting artifacts the discriminator has not learned to penalize.

Feedback manipulation. In systems that use observed attacker interactions as a learning signal, an adversary can deliberately interact in misleading ways to corrupt the generator's adaptation.

These risks reinforce a broader point: HoneyGAN deployments do not bring the arms race to an end. They simply shift its surface. The defender updates the discriminator; the generator adapts; the attacker develops a new detection method; and the cycle repeats. The hope is that the cycle now runs at machine speed on the defender's side, narrowing the gap that previously favored attackers.

## VI. EVALUATION AND VALIDATION APPROACHES

Evaluation is the bridge between an interesting research idea and an operational technology. This section sketches frameworks that, in our view, are necessary if GAN-generated honeypots are to move from prototype to production.

### *A. Detection-Resistance Testing*

The objective here is to determine whether generated honeypots can evade current detection tools. A reasonable methodology has three steps.

1. Tool collection. Assemble a representative set of publicly available honeypot detectors — for example, Honeypot Hunter and Send-Safe Honeypot Hunter — together with custom detection scripts developed by the research community and, where licensing permits, commercial scanning tools.
2. Test dataset preparation. Build three tiers of test data: real system configurations as a baseline, traditional honeypot configurations as a control, and GAN-generated configurations as the experimental group.
3. Evaluation protocol. Present each configuration to each detection tool, record detection rates and confidence scores, compute false-positive rates against the genuine baseline, and compare detection rates across the three tiers.

The two headline measures are detection rate (the fraction of decoys flagged by the detector) and evasion rate (its complement). Statistical significance testing is important here, since differences in detection rates can otherwise be over-interpreted.

### *B. Realism Assessment*

A second objective is to quantify how closely generated configurations match the distribution of real systems. This can be done with statistical tools — distance measures between distributions, divergence-based metrics, and discriminative tests trained on held-out data [16] — and with human evaluation, in which expert analysts attempt to distinguish real configurations from generated ones, ideally under blinded conditions.

### *C. Comparative Analysis Against Traditional Honeypots*

To benchmark GAN-generated honeypots against traditional approaches, the experiment must hold as much as possible constant. A fair design deploys roughly equal numbers of traditional and GAN-generated honeypots, places them in matched network locations with comparable exposure, monitors them continuously over an extended period (typically 30 to 90 days), and collects identical interaction metrics from both groups. Differences in those metrics — engagement rates, time-to-detection, intelligence yield — then reflect the underlying difference in approach rather than artifacts of placement.

### *D. Case Studies and Experimental Results*

The HoneyGAN Pots study by Gabrys and colleagues [8] provides early empirical support for the methodology. Their evaluation generated configurations for thirty distinct network devices, compared the resulting decoys against standard honeypot detection tools, and assessed both detection resistance and structural realism.

The findings are consistent with the theoretical argument made earlier in this paper. The GAN-generated decoys evaded existing honeypot detection systems noticeably more often than static configurations did. The generated configurations matched the statistical properties of real network devices closely enough that, after training converged, the discriminator could no longer reliably separate them from genuine systems. Both the OS-conditional and service-conditional variants produced targeted decoys that respected their conditioning while preserving overall realism. And once trained, the generator produced hundreds of unique configurations with little additional compute.

The authors are also careful to note the limits of the study: a relatively small evaluation of thirty devices, limited diversity in the training data, and no long-term observation of attacker behavior. Those caveats matter, and they map directly onto the open evaluation questions discussed in Section 5.4.

## VII. FUTURE RESEARCH DIRECTIONS

The space of GAN-generated honeypots is young, and there is room for substantial work on several fronts.

### *A. Integration with Reinforcement Learning*

As they stand, GANs are good at generating realistic configurations, but they do not learn from live attacker interactions. A natural next step is to combine them with reinforcement learning (RL) [19] into what one might call autonomous cyber-deception units. In such a system, the GAN component generates initial decoys; the RL component observes how attackers engage with those decoys, treats the engagement as a reward signal, and updates a policy that conditions future generation. The architecture is a closed loop:

Attacker interaction → reward signal → RL policy update → GAN conditioning → new decoy configuration → attacker interaction.

The appeal is that the system can move beyond static notions of realism toward effectiveness — generating the decoys that, in practice, attract and hold attackers and yield the most useful intelligence.

### *B. Blockchain for Verification and Audit Trails*

Operationally, security teams often struggle to distinguish among legitimate user activity, attacker activity in production systems, and attacker activity inside honeypots. Blockchain-based logging [20] offers one path to a clean audit trail.

Three integration points are worth investigating. Decoy registration anchors each generated configuration on a blockchain with a timestamp and a cryptographic hash. Legitimate-activity logging records authorized user interactions in the same store, providing a baseline. A verification protocol allows any future network event to be checked against the registry to determine whether it involved a registered decoy or a genuine system. The benefits are a tamper-resistant audit trail, a clear distinction between production and decoy environments, forensic evidence with strong provenance properties, and distributed verification without a single trusted authority.

### *C. Lightweight GANs for Edge and IoT Deployment*

GAN training and inference are computationally expensive, which limits where the technology can run. Specialized lightweight architectures aimed at edge and IoT deployment would broaden the deployment surface considerably.

| Technique | Description | Target setting |
|---|---|---|
| Knowledge distillation | Train a small student network from a larger teacher GAN [21]. | Inference-only devices |
| Pruning and quantization | Reduce model size and numerical precision [27]. | Resource-limited hardware |
| Federated learning | Distribute training across edge devices [22]. | Privacy-preserving adaptation |
| Hardware acceleration | Use specialized inference chips. | Low-latency requirements |

Plausible application areas include smart-home IoT honeypots, industrial control system decoys, mobile-device deception, and vehicular network security in autonomous platforms.

### *D. Multi-Agent Deception Ecosystems*

Today, individual honeypots tend to operate in isolation. There are gains to be had from deploying GAN-generated honeypots as a coordinated ecosystem that presents a consistent, internally referential narrative to an attacker.

Several patterns are worth exploring. Network-level consistency means generating decoys that reference one another — a fake database server that appears to back a specific fake web application, for instance. Behavioral synchronization means producing coordinated patterns of synthetic user activity across the deceptive environment. Progressive engagement means designing decoys that escalate the depth of interaction as an attacker invests more in them. And distributed intelligence capture means using multiple decoys across an environment to follow the different phases of an extended attacker campaign. Architecturally, this points toward multi-agent GAN designs in which several generators coordinate to produce coherent families of decoys rather than isolated artifacts.

### *E. Standardized Benchmarks and Evaluation Frameworks*

The lack of shared benchmarks makes it difficult to compare results across studies and slows the field's overall progress. A community-driven effort to standardize datasets, detection tooling, and metrics would change that. Useful components include reference datasets of network configurations and device fingerprints, annotated honeypot interaction logs, a

curated suite of state-of-the-art detection tools, automated and reproducible evaluation pipelines, and metric frameworks that cover security effectiveness, resource efficiency, and deception quality. Periodic competitions — DARPA-style challenges with public leaderboards — would help focus community attention on the open problems.

### *F. Ethical Considerations and Dual-Use Risks*

The literature on AI-driven deception is comparatively thin on ethics [23], and that gap should not persist. Several questions are worth addressing seriously rather than in passing.

- Dual use. The same generative techniques that build defensive decoys can be turned to offense — credible fake login pages, deepfake-driven phishing, environments that complicate forensic investigation.
- Proportionality. At what point does deception aimed at attackers begin to affect legitimate users in ways that are not justified by the security benefit?
- Transparency. Should organizations be required to disclose their use of AI-generated deception, and if so, to whom and under what conditions?
- Accountability. Where does responsibility lie when AI-generated decoys cause unintended harm — for example, when legal arguments about entrapment arise, or when legitimate users are caught up in a deception meant for someone else?

A workable governance framework would combine deployment guidelines, transparency standards, oversight mechanisms, international norms, and technical safeguards intended to prevent obvious misuse.

## VIII. CONCLUSION

This paper has argued for the use of Generative Adversarial Networks to construct a new kind of honeypot, and for treating cyber deception as a contest between two AI systems rather than as a static defensive artifact.

We began by examining the limits of traditional honeypots. They remain useful, but they are static, fingerprintable, and difficult to scale. They give attackers an asymmetric advantage: time. Any deception that hopes to keep pace with current threats has to be dynamic and largely automated.

Against that backdrop, we presented HoneyGAN as a concrete proposal. We described how a generator and discriminator can be trained to produce network device configurations that are statistically similar to real systems, and we walked through the conditional variants that allow defenders to ask for decoys with specific characteristics. We also surveyed the operational advantages: dynamic deception aligned with moving-target defense, automation-driven scalability, realism rooted in learned distributions, adaptability to new attack strategies, and more efficient resource allocation across the honeypot spectrum.

We were also explicit about the cost. GAN training is expensive. Mode collapse and instability are real risks, particularly in adversarial settings where homogeneous outputs are immediately exploited. Data is hard to come by. Evaluation metrics borrowed from the image-generation literature do not translate cleanly. And the same systems that generate decoys can be targeted directly through poisoning, evasion, or feedback manipulation.

Looking forward, several research directions seem particularly promising: combining GANs with reinforcement learning so that decoys adapt based on observed attacker behavior; using blockchain to maintain immutable audit trails over generated decoys; building lightweight architectures suitable for edge and IoT deployment; designing multi-agent ecosystems that present coherent deceptive environments rather than isolated decoys; standardizing benchmarks and evaluation; and developing ethical frameworks before broad deployment, rather than after.

More broadly, the HoneyGAN concept points toward several shifts in how we think about defense. From reactive to proactive: rather than waiting for an alarm, we shape the attacker's experience from the moment they begin reconnaissance. From static to dynamic: persistent change becomes the default rather than the exception. From human-scale to machine-scale: AI is used not only to analyze data but to generate defensive artifacts at a rate and volume no manual process can match. From an ad hoc arms race to a more explicit game-theoretic framing: the conflict is treated as a contest between intelligent agents, with the analytical tools that come with that view.

The threat landscape will continue to evolve, and attackers will continue to fold AI into their workflows. Defenders need to do the same — not only to detect attacks after the fact, but to actively misdirect and engage adversaries in real time. GAN-generated honeypots are one step toward that goal. If

reconnaissance is currently the attacker's greatest advantage, deception that consistently looks like the real thing erodes it: every minute spent on a decoy is a minute not spent on a real asset, and every interaction with a decoy is data for the defender.

The challenges we have laid out are real, and we do not expect them to dissolve quickly. But the alternative — defending tomorrow's networks with yesterday's static decoys — is worse. Realizing this vision will require sustained collaboration among security practitioners, machine-learning researchers, and systems architects. The foundations are in place. The harder, more interesting work is the one that lies ahead.

REFERENCES

[1] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative adversarial nets. Advances in Neural Information Processing Systems, 27.
[2] Spitzner, L. (2003). Honeypots: Tracking Hackers. Addison-Wesley Professional.
[3] Mirza, M., & Osindero, S. (2014). Conditional generative adversarial nets. arXiv preprint arXiv:1411.1784.
[4] Arjovsky, M., Chintala, S., & Bottou, L. (2017). Wasserstein generative adversarial networks. International Conference on Machine Learning, 214–223.
[5] Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434.
[6] The Honeynet Project. (2004). Know Your Enemy: Learning About Security Threats. Addison-Wesley Professional.
[7] Provos, N. (2004). A virtual honeypot framework. USENIX Security Symposium.
[8] Gabrys, R., Silva, C., & Bilinski, M. (2024). HoneyGAN Pots: Using generative adversarial networks to generate honeypot configurations. 2nd International Workshop on Adaptive Cyber Defense.
[9] Ndayipfukamiye, E., et al. (2025). Generative adversarial networks for adversarial defense in cybersecurity: A systematic review. Journal of Cybersecurity Research.
[10] Zhang, L., et al. (2024). MMHP-GAN: Mimicry honeypot feature generation using generative adversarial networks. Chinese Journal of Network and Information Security.
[11] Biggio, B., & Roli, F. (2018). Wild patterns: Ten years after the rise of adversarial machine learning. Pattern Recognition, 84, 317–331.
[12] Jajodia, S., et al. (2011). Moving Target Defense: Creating Asymmetric Uncertainty for Cyber Threats. Springer.
[13] Holz, T., et al. (2005). Measuring and detecting fast-flux service networks. Network and Distributed System Security Symposium.
[14] Salimans, T., et al. (2016). Improved techniques for training GANs. Advances in Neural Information Processing Systems, 29.
[15] Miyato, T., et al. (2018). Spectral normalization for generative adversarial networks. International Conference on Learning Representations.
[16] Gretton, A., et al. (2012). A kernel two-sample test. Journal of Machine Learning Research, 13(1), 723–773.
[17] Kurakin, A., Goodfellow, I., & Bengio, S. (2016). Adversarial machine learning at scale. arXiv preprint arXiv:1611.01236.
[18] Papernot, N., et al. (2016). Towards the science of security and privacy in machine learning. arXiv preprint arXiv:1611.03814.
[19] Mnih, V., et al. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529–533.
[20] Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. Decentralized Business Review.
[21] Hinton, G., Vinyals, O., & Dean, J. (2015). Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531.
[22] McMahan, B., et al. (2017). Communication-efficient learning of deep networks from decentralized data. Artificial Intelligence and Statistics, 1273–1282.
[23] Floridi, L., & Cowls, J. (2019). A unified framework of five principles for AI in society. Harvard Data Science Review, 1(1).
[24] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training GANs (Inception Score)," in Proc. NeurIPS, 2016.
[25] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "GANs trained by a two time-scale update rule converge to a local Nash equilibrium (FID)," in Proc. NeurIPS, 2017.
[26] C. Dwork and A. Roth, "The algorithmic foundations of differential privacy," Foundations and Trends in Theoretical Computer Science, vol. 9, no. 3–4, pp. 211–407, 2014.
[27] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," in Proc. ICLR, 2016.
[28] T. Chen et al., "AR-GAN: Adversarially robust generative adversarial network for autonomous-vehicle defense," IEEE Trans. Intell. Transp. Syst., 2022.
[29] ENISA, "Threat landscape report: IoT and Industry 4.0/5.0 attack-surface trends," European Union Agency for Cybersecurity, Tech. Rep., 2023.