International Journal of Science, Strategic Management and Technology
Volume **02** Issue **05** May-**2026** | ISSN: **3108-1762 (Online)** | Impact Factor: 3.8
An International, Peer-Reviewed, Open Access Scholarly Journal Indexed in recognized academic databases

Beyond Static Decoys: A Position Paper on Reinforcement-Learning- Driven Honeygan Ecosystems with Blockchain-Anchored Audit Trails for Industry 5.0 Cyber Defense

Rohit Kumar
Department of Computer Science Institute of Information Technology & Management
Affiliated to Guru Gobind Singh Indraprastha University New Delhi, India
ORCHID : https://orcid.org/0009-0000-4506-7143

Email : Rohit.kumar.it@iitmipu.ac.in

Shallu Hassija
Department of Computer Application Echelon Institute of Technology, Faridabad Affiliated to Guru Gobind Singh Indraprastha University New Delhi, India
ORCHID : https://orcid.org/my-orcid?orcid=0009- 0002-5719-1496

Email : shalluhassija@eitfaridabad.co.in

https://doi.org/10.55041/ijsmt.v2i5.247
Cite this Article: Kumar, R. & Hassija, S. (2026). Beyond Static Decoys: A Position Paper on Reinforcement-Learning- Driven Honeygan Ecosystems with Blockchain-Anchored Audit Trails for Industry 5.0 Cyber Defense. International Journal of Science, Strategic Management and Technology, *02*(05). https://doi.org/10.55041/ijsmt.v2i5.247

Abstract—Recent work on Generative Adversarial Networks for honeypot generation, most notably the HoneyGAN Pots framework, has shown that decoys produced by a learned generator can evade existing honeypot detectors more reliably than hand-crafted ones. That result is encouraging, but it is only a first step. A trained generator still produces decoys that are static after training: it does not learn from how attackers actually behave once the decoys are deployed, it provides no tamper-proof record of what it generated, and it assumes a class of compute resources that does not exist on industrial edge nodes. This paper is a position paper that takes the future-work agenda outlined by Gabrys et al. and develops it into a concrete research design. We propose an integrated framework, HoneyGAN+, that couples a conditional GAN to a reinforcement-learning policy trained on observed attacker engagement, anchors every generated decoy on a permissioned blockchain to give responders a verifiable audit trail, and uses knowledge distillation together with federated learning to push inference onto resource-constrained Industry 5.0 endpoints. We further argue that individual decoys should be replaced by coordinated multi-agent ecosystems whose internal references are mutually consistent, and we propose a standardized evaluation pipeline (detection resistance, engagement, longevity, and intelligence value) to make claims comparable across studies. The paper also discusses dual-use risks and a governance model that, in our view, ought to be developed before broad deployment rather than after. We are explicit throughout: the contribution is a research design with enough technical detail to support implementation, not a fully built and benchmarked system.

Index Terms—Generative adversarial networks, honeypots, reinforcement learning, blockchain, federated learning, multi-agent deception, Industry 5.0, adversarial machine learning, position paper.

I. INTRODUCTION

Cyber deception has had something of a quiet renaissance in the last few years. Honeypots, once mostly the province of researchers studying botnets [15], are reappearing in production environments as defenders look for ways to slow down attackers who themselves rely heavily on automation. The most interesting recent development on the defensive side is the use of Generative Adversarial Networks (GANs) to produce decoy configurations that statistically resemble real systems and therefore resist fingerprinting better than hand-crafted decoys. Gabrys, Silva, and Bilinski demonstrated this concretely in their HoneyGAN Pots framework [1], and subsequent work — for example MMHP-GAN [2] and the systematic review by Ndayipfukamiye et al. [3] — has reinforced that the basic idea generalizes.

Even so, the GAN-only formulation has limits that the original work itself flagged. Once trained, a generator does not improve from experience in the field: an attacker may engage a decoy, give up after twenty seconds, and the generator never learns. The training data is sensitive and hard to share across organizations, which slows the field as a whole. There is no built-in record of which decoys were generated, when they were instantiated, or how they evolved —

which becomes awkward in incident response, in legal contexts, and in any environment with a serious audit requirement. And the compute footprint of GAN inference, while modest by modern standards, is still too heavy for the kind of edge devices that dominate Industry 5.0 deployments and smart-city perimeters.

Our position is straightforward. The future of GAN-driven deception is not in larger generators alone, but in the systems we build around them. Four directions need to be pursued together rather than in isolation: reinforcement learning, so the system improves from attacker behavior; blockchain anchoring, so every decoy has a verifiable provenance; lightweight and federated training, so the same approach can run at the edge; and multi-agent coordination, so the decoys form a coherent deceptive environment rather than a scatter of unrelated artifacts. We treat these four threads as a single research design and call the resulting framework HoneyGAN+.

A. What This Paper Is and Is Not

Before listing contributions, we want to be explicit. This is a position paper. We propose a framework, describe its components in enough detail to be implemented, and outline an evaluation methodology. We do not, in this paper, report empirical results from a deployed prototype. That framing is deliberate. Rushing to publish premature numbers in this area would do more harm than good
— both because attacker models are easy to game, and because the evaluation methodology itself is part of what we are trying to establish. Where we describe expected behavior of the system, we say so. Where we are proposing an experiment rather than reporting one, we say so.

B. Contributions

With that framing in place, the contributions of this paper are as follows. First, we synthesize the open problems left by the HoneyGAN Pots line of work and argue that four threads — reinforcement learning, blockchain audit, edge-capable inference, and multi- agent coordination — must be pursued together. Second, we describe a concrete architecture, HoneyGAN+, that integrates these four threads into a single closed-loop system, with enough detail at the level of component interfaces, state and action spaces, and ledger schema to support implementation. Third, we propose a standardized evaluation framework along four axes (detection resistance, engagement, longevity, intelligence value) intended to make results across studies comparable. Fourth, we describe an experimental protocol — with baselines, datasets, and metrics — for validating the system as future work. Fifth, we treat dual-use and governance issues as first-class concerns, since the same machinery that builds defensive deception is comfortable building offensive deception too.

C. Paper Organization

Section II reviews the foundations we build on: GANs in security, honeypot evolution, and the open problems left by HoneyGAN Pots. Section III describes the proposed HoneyGAN+ architecture, including a system view and the inter-component interfaces. Section IV develops the reinforcement- learning loop and presents the training algorithm as pseudocode. Section V presents the blockchain audit- trail layer and the ledger schema. Section VI addresses lightweight and federated deployment. Section VII develops the multi-agent ecosystem view. Section VIII proposes an evaluation framework and a concrete experimental protocol. Section IX discusses ethical and governance issues. Section X summarizes the open problems, and Section XI concludes.

II. BACKGROUND AND MOTIVATION

A. GANs in Cyber Defense:

Where We AreGoodfellow et al.'s original formulation of the GAN [4] has by now been adapted to a long list of security applications: adversarial example generation [5], synthetic data for intrusion-detection training [6], traffic synthesis for safe testing, and — recently — decoy generation for honeypots [1], [2]. The architecture is familiar. A generator G learns to map noise z (and, for conditional variants, a label y) to samples that resemble real data, while a discriminator D learns to separate real from generated samples. The two networks train against one another in a minimax game whose equilibrium is, in principle, a generator

whose samples are indistinguishable from the real distribution under D's criteria.

For honeypots specifically, the most useful variant is the conditional GAN [7]. Conditioning lets the defender ask for a decoy of a particular type — a Linux web server, a Windows file share, an industrial controller — rather than a generic configuration. This matters operationally: a generic decoy that looks vaguely real often fails the simple smell test of fitting into the surrounding environment.

B. What HoneyGAN Pots Established, and What It Did Not

The HoneyGAN Pots study [1] established three things that earlier work did not. It showed that a GAN can learn the joint distribution of network device configurations from a modest training corpus. It showed that conditional variants can produce targeted decoys without sacrificing realism. And it showed that the resulting decoys evade established honeypot detectors more often than hand-crafted decoys do.

What it did not establish — and what the authors openly note — is whether such systems work over time, at scale, on edge hardware, in coordinated deployments, and under attack on the GAN itself. Each of these is a non-trivial research question, and each is the subject of one of the sections that follow.

C. The Industry 5.0 Threat Surface

It is worth being explicit about the deployment environment we have in mind. Industry 5.0 brings together human operators, collaborative robots, dense IoT instrumentation, and tightly integrated supply chains. The attack surface is heterogeneous in a way that earlier industrial environments were not: a single facility may run legacy SCADA protocols alongside modern container workloads, with edge gateways translating between them. ENISA and similar bodies have repeatedly noted that the threat surface in such deployments grows faster than defensive tooling adapts [8]. This is exactly the kind of environment in which static, manually configured honeypots fail quietly. Defenders simply do not have the time or staff to maintain them at the rate the environment evolves.

III. THE HONEYGAN+ ARCHITECTURE

HoneyGAN+ is best understood as a stack of four loosely coupled components rather than a monolithic system. Each component can be deployed independently, but the value comes from their combination. We describe the components first, the data flows between them second, and then summarize the architecture at the level of a single high-level view.

A. Component View

Generator (G). A conditional GAN that maps a noise vector z and a conditioning vector y — operating system, service profile, criticality tier, and a context vector summarizing the surrounding network — to a synthetic device configuration. We adopt a Wasserstein formulation with gradient penalty [9] for training stability, since mode collapse in this setting is operationally catastrophic.

Discriminator (D). Plays the standard adversarial role during training. After deployment, D continues to be useful as an internal sanity check: when retraining, the latest D is the first filter on whether new generated samples have drifted from realism.

Reinforcement-Learning Policy (π). Takes as input the current state of the deceptive environment

— which decoys are deployed, what attacker engagement has been observed, and current threat- intelligence signals — and outputs a request to G: what type of decoy to generate next, where to place it, and when to retire an existing decoy. The reward function combines engagement depth, time-to- detection, and intelligence yield.

Blockchain Ledger (L). A permissioned ledger whose state records, for each decoy, a cryptographic hash of the configuration, a timestamp of generation, the conditioning vector used, and a reference to the originating node. The ledger does not store the configurations themselves; it stores commitments to them.

B. Data Flow

At deployment time, the policy π emits a request
specifying the type and placement of a new decoy.

International Journal of Science, Strategic Management and Technology
Volume **02** Issue **05** May-**2026** | ISSN: **3108-1762 (Online)** | Impact Factor: 3.8
An International, Peer-Reviewed, Open Access Scholarly Journal Indexed in recognized academic databases

The generator G produces a configuration sample. A hash of the sample is committed to the ledger L, and the configuration itself is materialized — either as a low-interaction emulator or as a full virtual machine, depending on the criticality tier. Attacker interactions with the decoy are logged locally and summarized into a state update for π. Periodically, π is retrained against accumulated experience, and G is fine-tuned against fresh real-world configurations and against the latest D.

The architecture deliberately separates concerns. The generator does what GANs are good at: producing realistic samples. The policy does what reinforcement learning is good at: optimizing long- horizon reward. The ledger does what blockchains are good at: providing a tamper-evident sequence of commitments. None of these components is novel in isolation; the contribution lies in fitting them together for a specific defensive purpose.

C. System View

Figure 1 summarizes the architecture at a single glance. The policy π queries the current environment state, issues a generation request to G, which produces a configuration that is committed to L and then materialized into the network. Attacker telemetry flows back into the state, closing the loop. The federated training plane (Section VI) is shown as a separate annotation, since it operates on a slower timescale than the deployment loop.

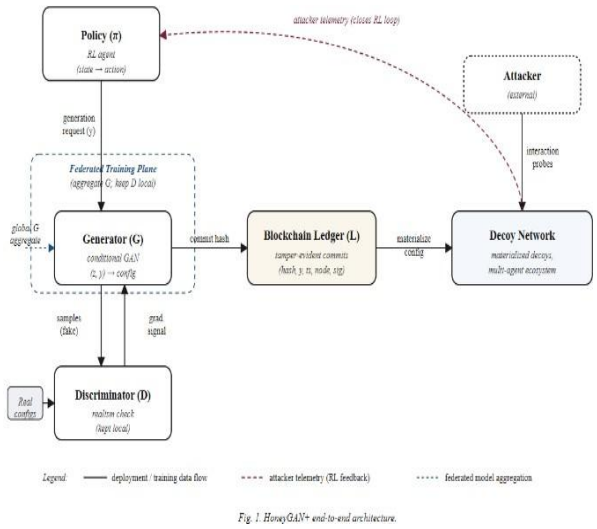

Fig. 1. HoneyGAN+ end-to-end architecture.

Fig. 1. HoneyGAN+ end-to-end architecture. Policy π drives generation; generator G produces configurations; ledger L records commitments; attacker telemetry feeds back to π. Federated training (dashed) operates on a slower cadence.

IV. REINFORCEMENT-LEARNING-DRIVEN ADAPTATION

A. Why Reinforcement Learning

A pure GAN approach optimizes for statistical realism. That is necessary but not sufficient. A decoy that looks perfectly real but never engages attackers is a poor decoy; one that engages them briefly and yields no useful telemetry is only marginally better. What the defender actually wants is decoys that maximize useful attacker engagement over time — a long-horizon objective, with a delayed and partially observed reward signal. This is precisely the setting reinforcement learning was designed for [10].

B. State, Action, and Reward

We define the policy's state s_t as a structured representation of the deceptive environment at time t: the set of currently active decoys with their conditioning vectors, a summary of recent attacker engagements (counts, depths, dwell times), and an exogenous threat-intelligence vector capturing current campaign indicators. The action space is discrete and structured: generate a new decoy with conditioning y, retire decoy d, regenerate decoy d, or take no action. The reward at each timestep combines three positive terms — engagement depth (how far into the deceptive environment an attacker progressed), intelligence yield (whether new TTPs were observed), and longevity (how long the decoy survived without being identified) — with weights tuned per deployment.

More formally, we write $r_t = \alpha \cdot e_t + \beta \cdot i_t + \gamma \cdot \ell_t - \delta \cdot h_t$, where e_t is engagement depth, i_t is intelligence yield, ℓ_t is the per-step longevity contribution, and h_t is a homogeneity penalty that discourages the policy from over-producing a single decoy type. The weights α, β, γ, δ are deployment- specific and, in the protocol described in Section VIII, are themselves a subject of evaluation.

C. Training Algorithm

Training the RL policy purely online is impractical: the reward signal is too sparse and too slow. We propose a hybrid scheme. The policy is bootstrapped on a simulator that replays historical honeypot interaction logs against a parameterized

attacker model, using offline policy-gradient methods for the initial training phase. After deployment, the policy is fine-tuned online using a more conservative algorithm — soft actor-critic or a constrained policy- gradient variant — to avoid catastrophic forgetting and to limit the rate of behavioral change in the deceptive environment, which would itself become a fingerprint.

Algorithm 1 below gives the overall training loop. We use the offline-then-online structure deliberately: the offline phase establishes a reasonable baseline policy before any real attacker interaction takes place, which we view as a safety property rather than just an efficiency one. An RL policy controlling deception in a production network should not be in a high-exploration regime when it first comes online.

Algorithm 1: HoneyGAN+ Policy Training Loop

Input: pretrained generator G, discriminator D,
offline log corpus L_off, attacker simulator A_sim,
environment Env, ledger L
Output: trained policy π

Initialize policy π with random weights
// Offline bootstrap phase
for epoch = 1 to N_offline do
Sample trajectory τ from A_sim using L_off
Compute returns R_t along τ using r_t formulation
Update π via policy gradient on (s_t, a_t, R_t)
end for

// Online deployment phase
while deployed do
Observe state s_t from Env
Sample action a_t ~ π(· | s_t)
if a_t = GENERATE(y) then
config c ← G(z, y)
h ← Hash(c); Commit(L, h, y, timestamp, node_id)
Deploy c into Env
else if a_t = RETIRE(d) then
Tear down decoy d; Commit(L, retire, d, timestamp)
end if
Observe reward r_t and next state s_{t+1}
Store (s_t, a_t, r_t, s_{t+1}) in replay buffer B
if t mod K = 0 then
Update π using SAC over batch from B
Apply KL-trust-region constraint to limit drift
end if
end while

D. Non-Stationarity and Diversity

There is a subtlety here that matters in practice. The attacker is not a stationary environment. As the deceptive environment changes, attacker behavior changes, and the reward distribution shifts. A naive RL agent may chase short-term reward in ways that compromise long-term deception — for instance, by repeatedly generating a decoy type that engaged a recent attacker, only to find that the resulting homogeneity becomes its own fingerprint. We mitigate this with two mechanisms: the explicit homogeneity penalty h_t in the reward, and an entropy regularizer on the action distribution. Together they bias the policy toward maintaining a diverse, plausible-looking environment even when a single decoy type seems to be performing well in the short term.

V. BLOCKCHAIN-ANCHORED AUDIT TRAILS

A. The Forensic Problem

Suppose an incident occurs and an analyst is reviewing a packet capture from three weeks ago. They see traffic to host 10.20.30.40 on port 22. Was that host a real production server, a decoy generated by HoneyGAN+, or — and this is the case that really matters — was it a real production server at the time of the incident, but is now a decoy? Without a tamper- evident record of what was deployed when, the analyst cannot answer reliably. Manual logs are not sufficient: they are exactly the kind of artifact an attacker on the same infrastructure would tamper with.

B. Ledger Design and Schema

We propose a permissioned blockchain — a Hyperledger Fabric or similar consortium ledger is appropriate — operated by the security team. Each generated decoy results in a transaction containing a cryptographic hash of the full configuration, the conditioning vector, the deployment timestamp, the

deployment location (network segment and asset identifier), and a signature from the generating node. Configurations themselves are stored off-ledger; the ledger commits to them rather than holding them. Retirements are recorded as well, so a decoy's lifecycle is fully reconstructable.

Concretely, each decoy lifecycle event is recorded as a transaction with the following fields: decoy_id (UUID), event_type (CREATE / UPDATE / RETIRE), config_hash (SHA-256 of the materialized configuration), conditioning_vector (the y passed to G), network_segment, asset_identifier, timestamp (UTC, monotonic at the issuing node), policy_version (the version of π that issued the request), and node_signature. The transaction is signed by the generating node and validated by the consortium according to the ledger's endorsement policy.

Three properties follow from this design. First, integrity: any subsequent attempt to retroactively claim that a particular host was always a decoy (or never was one) is detectable, because the ledger commits cannot be silently modified. Second, separation: legitimate user activity is anchored against a different namespace, so reconciliation between the two becomes a clean cryptographic check rather than an ambiguous log review. Third, accountability: when a deception causes unexpected operational impact — and this does happen — the chain of decisions that led to the deployment is on record, which both protects the security team and supports legitimate oversight.

C. Verification Protocol

At incident-response time, the analyst queries the ledger with the host identifier and the relevant time window. The ledger returns the set of lifecycle events for any decoy that matched that identifier during that window. The analyst then verifies, for each returned event, that the recorded config_hash matches the configuration recovered from forensic artifacts. A match means the host was a recognized decoy; the absence of any record during the window means the host was treated as production. This is a one-query operation in practice, and importantly, it remains valid even if the local logs on the host have been tampered with — the ledger is the authoritative record.

D. Cost and Practicality

It is fair to ask whether the additional complexity of a blockchain layer is justified. We think it is, for two reasons. The throughput required is modest — even a large enterprise generating thousands of decoys per day is well within the capacity of a permissioned chain — and the cost of a single major incident in which a decoy was mistaken for production (or vice versa) easily dwarfs the operational cost of running the ledger. We are not proposing a public chain; we are proposing a private ledger whose role is forensic, not financial.

VI. LIGHTWEIGHT AND FEDERATED DEPLOYMENT

A. The Edge Constraint

Industry 5.0 environments push compute outward. Edge gateways, programmable logic controllers, and IoT concentrators are increasingly the perimeter where deception needs to live. These devices typically have on the order of one to four gigabytes of memory and CPUs without modern accelerators. A full HoneyGAN inference path does not fit comfortably on such hardware, and training certainly does not.

B. Distillation for Inference

We propose a teacher-student architecture in the style of Hinton et al. [11]. A high-capacity teacher generator is trained centrally on the full training corpus. A much smaller student generator is trained to mimic the teacher's output distribution, using both a standard distillation loss on intermediate activations and an additional realism loss against the discriminator. The expected design point — informed by prior work on compressing generative models for edge deployment [22] — is a student model of roughly one to five percent of the teacher's parameter count. Whether such a student retains acceptable evasion performance against off-the-shelf honeypot detectors is an open empirical question, and one we take up explicitly in the evaluation protocol of Section

VIII. The student is what runs on the edge; the teacher remains in the central training plane.

C. Federated Training

Centralizing training data raises both privacy concerns (production configurations are sensitive) and regulatory concerns (data may not legally cross borders). Federated learning [12] offers a workable middle ground. Each participating organization trains a local generator on its own data and shares only model updates, not raw configurations. A central aggregator combines the updates into a global model, which is then redistributed. With differential-privacy noise added to the updates [13], the privacy guarantees become quite strong.

Federated training of GANs is harder than federated training of classifiers, because the adversarial dynamics interact awkwardly with aggregation. Recent work suggests that the FedAvg- style aggregation needs to be adapted — for instance, by aggregating only the generator and keeping discriminators local. We adopt that pattern here. The result is a generator whose realism benefits from the aggregate experience of all participants without any single participant's configurations leaving its perimeter.

VII. MULTI-AGENT DECEPTION ECOSYSTEMS

A single convincing decoy is good. A network of decoys that reference one another consistently is much better — and in fact closer to what a sophisticated attacker would expect of a real environment. Real production networks are not isolated islands of services; they are interlinked. A web server has a backend database. A jump host has access patterns that match the operations team. A file share has users who actually use it.

A. Coordinated Generation

We extend HoneyGAN+ to a multi-agent setting in which several conditional generators coordinate through a shared latent context. The context vector encodes facts about the deceptive environment as a whole — the dominant operating-system mix, the apparent business function of the segment, the time zone of synthetic users — and is consumed by every generator. The result is a population of decoys that, when probed, exhibit the kind of cross-references a real environment exhibits: the fake database server's logs reference the fake application server, the fake application server's configuration references the fake load balancer, and so on.

B. Synthetic User Activity

Decoys without users look exactly like decoys without users, which is to say, like decoys. Static honeypots have always struggled with this: real systems have logged-in sessions, mouse movements on RDP, scheduled tasks running quietly in the background. A second class of generative model — sequence models trained on benign user activity — can produce synthetic activity that is associated with the relevant decoys. The activity does not have to be elaborate, but it has to exist, and it has to fit the system it is on. A Linux database server with no SQL queries on it is a tell.

C. Progressive Engagement

Multi-agent ecosystems also enable progressive engagement, in which the depth of interaction available to an attacker scales with their apparent investment. An attacker who runs a single port scan gets the low-interaction surface; one who attempts authentication gets a more elaborate response that includes references to other (deceptive) infrastructure; one who pivots from there gets escalated into a full high-interaction environment whose internal coherence has been generated specifically for that engagement. This is the kind of strategy that pays off only when the underlying decoys can be generated and adapted at machine speed, which is exactly what HoneyGAN+ is meant to enable.

VIII. TOWARD A STANDARDIZED EVALUATION FRAMEWORK

One of the more frustrating things about reading the current literature on GAN-driven deception is that comparing results across studies is essentially impossible. Each paper picks its own evaluation, its own detection tools, and its own definition of success.

We are not the first to point this out — the systematic review by Ndayipfukamiye et al. [3] makes the same observation — and proposing a fix is overdue.

A. Four Axes

A credible evaluation of a HoneyGAN-style system needs to report on four distinct axes. Detection resistance measures whether existing honeypot detectors flag the decoy. Engagement measures whether attackers, having reached the decoy, interact with it as they would with a real system. Longevity measures how long the decoy operates before being identified, which is a different question from whether a one-shot detector catches it. Intelligence value measures whether the data captured during engagement is operationally useful — concretely, whether it improves downstream detection of real attacks against real assets.

B. Proposed Test Battery

TABLE I. PROPOSED EVALUATION TEST BATTERY

Axis	Method	Primary Metric
Detection Resistance	Run a curated set of public and licensed honeypot detectors against generated and baseline configurations.	Evasion rate (1 − detection rate)
Engagement	Deploy in a controlled red-team exercise against parameterized attacker profiles.	Mean interaction depth
Longevity	Long-running observation of attacker behavior over 30–90 days.	Median time-to-identification
Intelligence Value	Compare downstream detection improvement on held-out real attacks.	Lift in true-positive rate

This is a starting point, not a finished benchmark. Reasonable people will disagree on the weights, the attacker models, and the choice of detection tools. Our claim is more modest: that the field needs a shared starting point, and that any starting point is better than none.

C. Experimental Protocol for HoneyGAN+ (Future Work)

To make the evaluation framework concrete, we outline the protocol we intend to follow in the empirical phase of this project. We document this here, in advance of running the experiments, because we believe pre-registration of methodology is a healthy habit in security research — it reduces the temptation to retrofit hypotheses to results.

Baselines. We will compare HoneyGAN+ against three baselines: (i) standard low-interaction honeypots (Cowrie, Conpot); (ii) the original HoneyGAN Pots generator [1] without the RL, blockchain, or distillation layers; and (iii) a HoneyGAN+ variant with each layer ablated in turn, to isolate the contribution of each component.

Datasets and Environments. Training data for the generator will be drawn from publicly available device-configuration corpora and from sanitized institutional data where available. Attacker engagement will be measured first in a sandboxed environment populated by scripted attacker agents at three skill tiers (opportunistic scanning, targeted exploitation, persistent campaign), and subsequently in a controlled red-team exercise. We do not propose to deploy in production until both phases are complete.

Metrics. Each of the four axes in Table I has a primary metric, but in the protocol we also record secondary metrics: per-class evasion rate (detection resistance); action diversity entropy as a proxy for engagement quality; distribution of decoy lifetimes (longevity); and the precision-recall curve of downstream detectors trained on HoneyGAN+ telemetry versus on baseline honeypot telemetry (intelligence value). Each experiment will be repeated across at least five random seeds, and we will report mean and 95% confidence intervals rather than single-run numbers.

Honest reporting. We commit, in this protocol, to publishing the results of this evaluation regardless of outcome. A negative or partial result for any of the four layers of HoneyGAN+ is a useful contribution; concealing it would not be.

IX. ETHICAL AND GOVERNANCE CONSIDERATIONS

AI-driven deception sits in an uncomfortable spot ethically. The technology is genuinely defensive in motivation, but the same generators that build defensive decoys are equally capable of building offensive deception — convincing fake login portals for credential harvesting, deepfake-driven social engineering, or fabricated systems intended to entrap rather than to detect. The literature on this point is, to put it gently, thin [14].

A. Dual Use

There is no clean technical fix for dual use. A generator trained on legitimate configurations can produce decoys; the same generator can produce convincing artifacts for an attacker. What can be done is to make the operational use harder to misdirect: requiring transactions on the audit ledger, requiring approved conditioning vectors, and logging the full lifecycle of every generated artifact. None of this stops a determined operator inside an organization, but it makes casual misuse visible.

B. Proportionality and Transparency

Deception, by design, affects users who interact with it. Ideally, those users are attackers, but in practice some legitimate users encounter decoys too
— a misconfigured scanner, a vendor's vulnerability assessment, a curious employee. We argue that a proportionality principle should apply: deception should be no more elaborate than necessary; decoys should not be deployed in places where benign users are likely to land; and material harms to third parties should trigger immediate retirement. On transparency, the question is whether organizations should disclose the use of AI-driven deception. We think the honest answer is: yes, to regulators and incident responders, with appropriate confidentiality; not necessarily to potential attackers, since the disclosure would defeat the purpose.

C. Governance

Governance frameworks for AI-driven deception are still in their infancy. Floridi and Cowls's five- principle framework for AI in society [14] is a reasonable starting point, but it was not written with deception in mind, and its application to this domain needs work. We see four practical near-term steps: deployment guidelines that distinguish research from production; oversight mechanisms inside organizations that approve and review deception campaigns; international norms on what counts as proportionate use; and technical safeguards — including the audit ledger described in Section V — that make accountability mechanically enforceable.

X. OPEN PROBLEMS

We close with the open problems that we think matter most. The first is data: federated training for adversarial models is still genuinely difficult, and the lack of shared corpora is slowing the field. The second is evaluation: until there is a benchmark with broad community buy-in, claims about the relative merits of different approaches will be hard to credit. The third is robustness: GAN poisoning, discriminator evasion, and feedback manipulation are not theoretical; any production deployment will face them. The fourth is integration: HoneyGAN+ is a stack of components, and stacks have edges where things go wrong, especially under the operational pressure of an active incident.

None of this argues against the approach. It argues for taking it seriously enough to invest in the unglamorous parts. The original HoneyGAN Pots work showed that the core idea is sound. The work that remains is to make it real: robust enough to deploy, lightweight enough to run at the edge, transparent enough to govern, and coordinated enough that a deceptive environment looks like an environment rather than a collection of artifacts.

XI. CONCLUSION

Defense, historically, has been on the slow side of the AI arms race. The systems we have described here are an attempt to change that — not by matching attackers move for move, but by changing the game so that the attacker's reconnaissance advantage is eroded at the source. We have offered an architecture, a training algorithm, a ledger schema, and an evaluation protocol; we have not, in this paper, offered final empirical numbers, and we have been

explicit about that. Whether HoneyGAN+ delivers in practice is, ultimately, an empirical question, and we hope this paper helps frame the experiments that will answer it.

ACKNOWLEDGMENT

The author thanks colleagues at the Department of Computer Science, Institute of Information Technology and Management, New Delhi, for discussions on the operational realities of honeypot deployment that shaped the framing of this paper.

REFERENCES

[1] R. Gabrys, C. Silva, and M. Bilinski, "HoneyGAN Pots: Using generative adversarial networks to generate honeypot configurations," in Proc. 2nd Int. Workshop on Adaptive Cyber Defense, 2024.

[2] L. Zhang et al., "MMHP-GAN: Mimicry honeypot feature generation using generative adversarial networks," Chinese Journal of Network and Information Security, vol. 10, no. 2, pp. 45–59, 2024.

[3] E. Ndayipfukamiye et al., "Generative adversarial networks for adversarial defense in cybersecurity: A systematic review," Journal of Cybersecurity Research, 2025.

[4] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in Advances in Neural Information Processing Systems, vol. 27, 2014.

[5] B. Biggio and F. Roli, "Wild patterns: Ten years after the rise of adversarial machine learning," Pattern Recognition, vol. 84, pp. 317–331, 2018.

[6] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training GANs," in Advances in Neural Information Processing Systems, vol. 29, 2016.

[7] M. Mirza and S. Osindero, "Conditional generative adversarial nets," arXiv preprint arXiv:1411.1784, 2014.

[8] ENISA, "Threat landscape report: IoT and Industry 4.0/5.0 attack-surface trends," European Union Agency for Cybersecurity, Tech. Rep., 2023.

[9] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved training of Wasserstein GANs," in Advances in Neural Information Processing Systems, vol. 30, 2017.

[10] V. Mnih et al., "Human-level control through deep reinforcement learning," Nature, vol. 518, no. 7540, pp. 529–533, 2015.

[11] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," arXiv preprint arXiv:1503.02531, 2015.

[12] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. Aguera y Arcas, "Communication-efficient learning of deep networks from decentralized data," in Proc. AISTATS, pp. 1273–1282, 2017.

[13] 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.

[14] L. Floridi and J. Cowls, "A unified framework of five principles for AI in society," Harvard Data Science Review, vol. 1, no. 1, 2019.

[15] L. Spitzner, Honeypots: Tracking Hackers. Addison-Wesley Professional, 2003.

[16] The Honeynet Project, Know Your Enemy: Learning About Security Threats. Addison-Wesley Professional, 2004.

[17] N. Provos, "A virtual honeypot framework," in Proc. USENIX Security Symposium, 2004.

[18] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in Proc. ICML, pp. 214–223, 2017.

[19] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, "Spectral normalization for generative adversarial networks," in Proc. ICLR, 2018.

[20] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," self-published whitepaper, 2008.

[21] T. Chen et al., "AR-GAN: Adversarially robust generative adversarial network for autonomous- vehicle defense," IEEE Trans. Intell. Transp. Syst., 2022.

[22] 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.

[23] 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," in Advances in Neural Information Processing Systems, 2017.

[24] S. Jajodia, A. K. Ghosh, V. Swarup, C. Wang, and X. S. Wang, Eds., Moving Target Defense: Creating Asymmetric Uncertainty for Cyber Threats. Springer, 2011.

[25] N. Papernot, P. McDaniel, A. Sinha, and M. Wellman, "Towards the science of security and privacy in machine learning," arXiv preprint arXiv:1611.03814, 2016.