Write Pressure and Canary Drift: Monitoring and Guardrailing Inference-Time Learning for Long-Context Sequence Models

David Montgomery
Independent Researcher

January 2, 2026

Abstract

Inference-time learning and test-time training (TTT) convert part of deployment from fixed inference into online adaptation. This design can improve robustness under distribution shift [1, 2] and can compress long input streams into compact learned state, reducing reliance on large token buffers. These advantages come with a new security and reliability surface: untrusted inputs can become optimization targets that modify parameters or fast-adapting memory modules. Prompt injection already exploits the lack of a clean separation between instructions and data in LLM applications [3, 4]. Optimization-based attacks show that attackers can automatically search for strings that bypass safety behavior [5, 6]. When learning occurs at inference, these attacks can target not only outputs but also the update mechanism, creating poisoning-like dynamics.

This paper proposes a simple, composable guardrail loop for inference-time learning in sequence models: (i) *write pressure* monitoring that estimates how strongly an input attempts to update a learnable module via gradient norms, update norms, and per-token influence; (ii) *pre-update gating* that blocks or scales updates under suspicious statistical patterns, instruction-override heuristics, or an out-of-distribution confusion plus heavy-write criterion; and (iii) *canary drift rollback* that treats each update as a transaction and reverts updates that measurably degrade stability. We provide an open-source toy implementation as an evaluation harness and argue that defense in depth is necessary because adaptive attackers can satisfy one constraint while violating another. The framework prioritizes observability, auditability, and modular composition with existing prompt-injection defenses.

1 Introduction

Large language models are commonly deployed as static functions at inference time: model parameters are fixed, and adaptation to a user, task, or document occurs by conditioning on tokens within a context window. This paradigm is powerful but operationally constrained. Self-attention in Transformers provides strong modeling capacity [7], yet long contexts impose substantial compute and memory costs. Standard attention scales quadratically with sequence length, and autoregressive decoding requires storing key-value caches whose size grows with context length. Kernel-level optimizations such as FlashAttention improve throughput and memory efficiency by reducing memory traffic [8], but long-context inference remains resource-intensive.

A parallel trend explores architectures and mechanisms that reduce reliance on large token buffers. Structured state space models (SSMs) and recurrent hybrids aim to provide linear-time or constant-memory inference characteristics while maintaining competitive accuracy. Mamba uses selective state-space mechanisms [9]. RetNet proposes retention with parallel training and recurrent inference [10]. RWKV reformulates attention-like computation into a recurrent form [11]. Hybrid designs such as Jamba interleave Transformer and Mamba blocks and use mixture-of-experts to increase capacity [12, 13].

Separately, inference-time learning converts part of the model from static inference into online adaptation. Test-time training (TTT) shows that self-supervised updates at test time can improve robustness under

distribution shift [1]. Test-time adaptation methods such as Tent update a constrained set of parameters online via entropy minimization [2]. Parameter-efficient adaptation methods such as LoRA and QLoRA localize learning to low-rank modules [14, 15], making online updates operationally feasible.

Inference-time learning changes the security model. Prompt injection is consistently highlighted as a top risk in real-world LLM applications [3]. Indirect prompt injection, where external content contains malicious instructions, becomes especially relevant when retrieval or tools are used [4]. Optimization-based attacks show that automated search can discover injection strings and suffixes that bypass safety behavior, including in aligned systems [5, 6, 16]. When inference includes parameter updates, the attacker can target the update mechanism itself, pushing the system closer to online poisoning and backdoor dynamics [17, 18, 19].

This paper proposes a monitoring and guardrail loop designed for inference-time learning in sequence models. The goal is not a complete defense but a practical template that improves observability and supports defense in depth:

1. **Write pressure monitoring** estimates how strongly an input attempts to modify a learnable memory module.
2. **Pre-update gating** blocks or scales updates based on statistics, heuristics, and a confusion plus heavy-write rule.
3. **Canary drift rollback** treats each update as a transaction, reverting updates that degrade stability.

A toy open-source implementation accompanies the paper to provide a reproducible evaluation harness [20].

2 Background and related work

2.1 Transformers, long context, and sequence alternatives

The Transformer architecture uses self-attention to replace recurrence, enabling parallel training and strong performance across language tasks [7]. Standard attention has $O(n^2)$ compute in sequence length n, motivating both approximate attention and system-level improvements. FlashAttention improves exact attention speed and memory efficiency through IO-aware tiling [8]. These improvements help but do not remove long-context inference costs.

Non-attention architectures seek different scaling properties. Mamba is a selective SSM that aims to match or exceed Transformer performance while maintaining linear-time sequence modeling behavior [9]. RetNet introduces a retention mechanism with parallel and recurrent computation paths, targeting low-cost inference and long-sequence modeling [10]. RWKV combines transformer-like training parallelism with recurrent inference by reformulating computations to avoid explicit attention [11]. Jamba interleaves Transformer and Mamba layers and uses mixture-of-experts to increase capacity while keeping active compute manageable, reporting strong long-context performance [12, 13]. These works are relevant because they shift how context is represented at inference, creating space for learned state, recurrence, and adaptation mechanisms.

2.2 Inference-time adaptation, TTT, and fast weights

Test-time training updates model parameters at inference using a self-supervised objective derived from the test sample, improving robustness under distribution shift [1]. Tent performs fully test-time adaptation by minimizing prediction entropy, updating normalization parameters and affine transformations online [2]. These approaches emphasize two design constraints: restrict which parameters can change, and bound update magnitude.

The idea of rapid adaptation through weights has a long history. Fast weights provide rapidly changing weights that store recent information and implement attention to the recent past [21]. Meta-learning methods

such as MAML explicitly train parameters that can be adapted with a small number of gradient steps [22]. In language, meta-training approaches such as MetaICL and in-context tuning support test-time task adaptation through conditioning and learned instruction formats [23, 24]. Meta-in-context learning further explores how sequential contexts can recursively improve in-context learning itself [25]. These lines motivate systems that learn from their deployment inputs, whether via parameters or via structured memory.

Parameter-efficient fine-tuning methods localize adaptation to small modules. LoRA injects low-rank trainable matrices into a frozen base model [14]. QLoRA combines quantized base weights with low-rank adapters to reduce memory while maintaining performance [15]. These modules provide a narrow and auditable write surface that is attractive for inference-time learning.

2.3 Prompt injection, indirect injection, and optimization-based attacks

Prompt injection exploits the fact that LLMs do not reliably separate instructions from data. Security guidance such as OWASP lists prompt injection, insecure output handling, and training data poisoning as top LLM application risks [3]. Indirect prompt injection occurs when external content contains malicious instructions that are retrieved and appended to the prompt. BIPIA introduces a benchmark for indirect prompt injection and develops black-box and white-box defenses [4]. Authentication-style test-time defenses also appear, such as FATH, which uses formatting and hashing tags to separate policy and user instruction responses [26]. TopicAttack highlights more natural indirect injection by topic transitions [27]. System prompt poisoning extends the attack surface to system prompts, creating persistent effects beyond user-level injections [28].

Optimization-based prompt attacks show that automated search can discover effective injection strings. Universal and transferable adversarial suffixes can bypass alignment through greedy and gradient-based methods [5]. Automatic and universal prompt injection uses gradient-based search to generate injection data across models [6]. Subsequent work studies mechanisms of these attacks, including attention hijacking properties [16], and explores efficient black-box generation of adversarial suffixes [29]. These results motivate defenses that assume adaptive attackers and that avoid reliance on single heuristics.

3 Threat model for inference-time learning

3.1 System model

Consider a deployed sequence model with base parameters θ and a learnable memory module with parameters ϕ. Deployment freezes θ while allowing ϕ to update online. Updates occur while processing input chunks x_c:

$$\phi \leftarrow \phi - \eta \nabla_\phi \mathcal{L}(x_c; \theta, \phi), \tag{1}$$

where $\mathcal{L}$ is a self-supervised or auxiliary objective such as next-token prediction, entropy minimization, or contrastive learning. The memory module can be implemented as low-rank adapters, a hypernetwork, a restricted affine transformation, or a structured recurrent memory component.

This abstraction covers TTT-style update loops and adapter-based inference-time learning. It also covers recurrent and SSM architectures that include a small adaptive component that performs compression or adaptation.

3.2 Attacker model

An attacker can supply inputs that the system processes and may update on. Inputs can be direct user messages, indirect retrieved documents, or tool outputs. The attacker may not know θ or ϕ, yet transferability and black-box optimization make attacks plausible [5, 6, 29]. The attacker aims to cause harmful adaptation, degrade reliability, saturate the memory module, or induce policy drift.

3.3 Risk taxonomy

Inference-time learning introduces risks beyond ordinary prompt injection:

- **Session poisoning:** updates persist within a session and degrade subsequent behavior.
- **Backdoors and triggers:** triggers cause updates that later activate undesired behavior, analogous to backdoor attacks in prompt engineering [19].
- **Denial of learning:** adversarial inputs saturate the memory module and reduce useful adaptation.
- **Instability and drift:** updates cause loss of coherence or calibration.
- **Cross-boundary confusion:** the system learns from data content as if it were instruction content, paralleling indirect prompt injection [4, 27].

The central safety requirement is *write control*, meaning the system needs an explicit policy for when it is safe to learn, what to learn, and how much to learn.

4 Write pressure monitoring

Write pressure quantifies how strongly an input attempts to modify the learnable module.

4.1 Primary signals

For each chunk c:

- **Gradient norm:** $g_c = \|\nabla_\phi \mathcal{L}(x_c; \theta, \phi)\|_2$.
- **Update norm:** $u_c = \|\Delta\phi_c\|_2$, where $\Delta\phi_c$ is the applied parameter change.
- **Token influence proxy:** an attribution score from gradients with respect to input embeddings, used to identify which tokens contribute most to write pressure.

Gradient norm approximates the local sensitivity of the objective to the learnable module and reflects how strongly the optimizer wants to change ϕ on this chunk. Update norm reflects how much change actually occurred after optimizer constraints, learning rate, and clipping. Per-token influence highlights which tokens drive update pressure and supports forensic analysis.

4.2 Robust anomaly scoring

Absolute thresholds are brittle across domains. A practical alternative flags anomalies relative to recent history using robust statistics. A median and median absolute deviation (MAD) based robust z-score provides an outlier-resistant measure of whether g_c or u_c is unusually high in the current session. This paper treats robust scoring as a practical heuristic rather than a guarantee. Robust scoring is useful for operators because it creates adaptive thresholds that follow baseline drift.

4.3 Interpretation

Write pressure is a control signal, not a semantic classifier. High write pressure indicates that the model is surprised by the input and the optimizer wants to fit it. This can occur for benign inputs such as code, logs, or new jargon. It can also occur for adversarial reasons such as gibberish suffixes, blob injections, or crafted token patterns that target optimization dynamics [5, 16]. Monitoring write pressure provides observability and supports control policies, but it must be combined with additional signals to decide whether learning should be allowed.

5 Pre-update gating

Pre-update gating decides whether to apply an update for a given chunk. Conservative gating reduces the attack surface by preventing learning from suspicious inputs while still allowing forward inference to proceed.

5.1 Token statistics

Two cheap statistics are useful:

- **Token entropy:** Shannon entropy of the chunk token distribution, measured in bits per token.
- **Token diversity:** fraction of unique tokens in the chunk.

Very low entropy or diversity often indicates repetition, templated attacks, or non-natural input. These signals require calibration on expected workloads because legitimate code and logs can also reduce token diversity.

5.2 Blob and instruction-override heuristics

Practical systems can include lightweight detectors for base64-like tokens, long hex strings, minified code blobs, and explicit instruction-override templates. These heuristics do not provide robustness under adaptive attackers. Their value is in blocking common patterns and reducing accidental learning from obviously non-natural input. Indirect injection benchmarks suggest that content often contains instruction-like substrings, so heuristics should be conservative about learning from any content that resembles instruction override [4].

5.3 Out-of-distribution plus heavy write

A key rule blocks learning when the model is both confused and attempting to learn. Confusion is estimated by the self-supervised loss on the chunk. The gate blocks updates when the chunk loss exceeds a threshold and the gradient norm exceeds a smaller threshold:

$$\text{block if } \mathcal{L}_c > \tau_{\text{loss}} \text{ and } g_c > \tau_{\text{grad}}. \tag{2}$$

This captures a common failure mode where non-linguistic blobs produce high loss while still creating non-trivial write pressure. The criterion is motivated by a poisoning perspective: a confused model that writes is vulnerable to being steered by out-of-distribution patterns.

The idea of using loss-based and outlier-based criteria connects to poisoning literature. Poisoning attacks craft training points to increase validation loss [17]. Certified defenses examine worst-case loss increases under poisoning, often relying on outlier removal or robustness assumptions [18]. In prompt space, optimization-based suffix attacks often occupy atypical regions while remaining effective [5]. The OOD plus heavy write criterion is a lightweight defense-in-depth rule that catches cases where a single gradient threshold can be satisfied while the input remains highly out-of-distribution.

5.4 Learn scaling

Binary allow or block decisions are useful for conservative baselines. Practical deployments benefit from *learn scaling*. Suspicious chunks can be processed with reduced learning rate, reduced rank, or learning disabled while still allowing inference to proceed. Learn scaling reduces false positives on code and logs while keeping a strong response for highly suspicious blobs or injection templates.

6 Canary drift and rollback

Pre-update gating reduces risk but does not eliminate it. Post-update checks catch subtle cases where an update was allowed but caused negative effects.

6.1 Canary suite

A canary suite is a fixed set of probes designed to remain stable under benign updates. A simple coherence canary is a short text snippet for which next-token prediction loss should remain stable. More robust deployments include:

- **Coherence canaries:** language modeling loss on benign text.
- **Policy canaries:** refusal invariance and safety policy adherence.
- **Tool-use canaries:** constraints on tool calling and output handling, aligned with LLM application security concerns [3].

This paper focuses on coherence canaries to remain model-agnostic.

6.2 Rollback as a transaction

Rollback treats each update as a transaction:

1. Snapshot ϕ_{t-1}.
2. Apply update to obtain ϕ_t.
3. Evaluate canary score $s(\phi)$ before and after.
4. Revert to ϕ_{t-1} when the canary delta exceeds a threshold.

Rollback resembles validation-loss step rejection and aligns with continual learning constraints that preserve previously learned behavior [30]. Rollback also provides an auditable safety mechanism: an operator can observe that an update was attempted, evaluated, and rejected.

6.3 Rollback thresholds

Two practical thresholds are useful:

- **Absolute canary delta:** revert when $s(\phi_t) - s(\phi_{t-1})$ exceeds a fixed threshold.
- **Robust z-score delta:** revert when the canary delta is an outlier relative to recent deltas.

The robust threshold adapts to baseline noise, while the absolute threshold prevents large drift even when history is short.

7 Toy implementation and artifact

We provide a toy implementation as an educational sandbox and a reproducible evaluation harness [20]. The implementation freezes a small GRU-based base model and allows only a linear adapter module to update at inference. The monitor computes gradient norms, update norms, per-token influence, and robust anomaly scores. The gate uses token entropy and diversity, blob and instruction-override heuristics, and the OOD plus heavy write rule. A lightweight dashboard visualizes write pressure, loss, canary deltas, and gating decisions.

Table 1: Defense in depth outcomes in the toy sandbox. The OOD plus heavy write rule blocks updates even when a simple gradient norm constraint is satisfied.

Scenario	Grad norm	Loss	Gate outcome
Benign text	low	moderate	allow
Instruction override template	moderate	moderate	block via heuristic
High-entropy blob	high	high	block via OOD plus heavy write
Optimized low-grad blob	near threshold	high	block via OOD plus heavy write

The toy model intentionally uses a one-way hashed token-to-id mapping. This makes it explicit that the monitor targets optimization dynamics rather than English semantics. It also surfaces a practical detail: attack tooling that attempts to generate human-readable strings must operate over the real tokenizer and vocabulary. A one-way mapping prevents inversion, which is useful for demonstration but not representative of deployed tokenizers.

8 Experimental evaluation

The evaluation goals are:

- Demonstrate that write pressure flags out-of-distribution blobs.
- Demonstrate that a single gradient threshold is insufficient as a gate.
- Demonstrate that defense in depth blocks multi-objective optimization attempts that satisfy one constraint while violating another.

8.1 Scenario families

We consider three scenario families:

1. **Benign text:** ordinary English-like chunks.
2. **Instruction override templates:** explicit injection strings similar to common prompt injection patterns and indirect injection benchmarks [3, 4].
3. **High-entropy blobs:** base64-like repeated tokens and long hex-like strings.

8.2 Illustrative outcomes

The toy monitor reproduces a typical pattern for blobs: early chunks yield high loss and high gradient norms, meaning the model is confused and tries to learn the blob. If learning is allowed, write pressure often decreases as the model memorizes the pattern, which is the same dynamic observed in many learning systems. If learning is blocked, loss remains high but updates remain near zero, avoiding adapter pollution.

A synthetic red-team attempt that optimizes inputs to keep g_c below a single threshold can still be blocked by the OOD plus heavy write rule because $\mathcal{L}_c$ remains high while g_c stays above a smaller threshold. This reflects a general lesson from optimization-based attacks: adaptive attackers can satisfy one constraint while still causing harm, so multi-signal gating is more robust than a single metric [5, 6, 16].

9 Discussion

9.1 Learning as a capability surface

Inference-time learning creates a writeable surface. The operational decision becomes when to allow writes, how to bound them, and how to detect and undo harmful writes. Write pressure monitoring turns a hidden optimization process into an observable control signal. Pre-update gating restricts when writes are allowed. Canary rollback provides a post-update safety net when pre-update gates fail or when an update has unexpected side effects.

9.2 False positives and learn scaling

Hard blocking can create false positives on legitimate but unusual inputs such as code, logs, and compressed text. Learn scaling provides a smoother policy: suspicious chunks can be processed with reduced learning rate, reduced rank, or learning disabled, while still allowing inference. Calibration requires baseline measurements on representative workloads and should be reported as part of evaluation.

9.3 Relationship to prompt injection defenses

Prompt injection defenses often attempt to separate instructions from data. Inference-time learning adds a second axis: separation of content that is safe to learn from from content that should be read but not used as an optimization target. This reframes part of prompt injection as an online poisoning problem, connecting to poisoning and robustness literature [17, 18]. It also supplies a clean interface for system design: a system may read untrusted content to answer a question while refusing to treat that content as training signal.

9.4 Canary design and policy drift

Coherence canaries detect gross corruption but do not enforce policy. Policy and tool-use canaries broaden coverage and align with security taxonomies such as OWASP [3]. Persistent prompt poisoning risks, including system prompt poisoning, motivate canary suites that explicitly test invariants beyond coherence [28]. Canary suites should be versioned, audited, and treated like regression tests.

10 Limitations and future work

This work provides a conceptual and practical template rather than a proof of security. Limitations include:

- Write pressure can be gamed, so robust systems require multi-signal monitoring and adversarial evaluation against adaptive attackers.
- Heuristic gates are brittle and require maintenance.
- Coherence-based canaries are coarse proxies and benefit from policy and tool-use canaries.
- Real deployments require session isolation to prevent cross-user persistence, especially in systems that cache learnable modules.

Future work includes applying these ideas to open-weight LLMs with real tokenizers, benchmarking false positive rates on code and logs, integrating policy canary suites, and exploring principled update constraints inspired by continual learning, robustness, and certified poisoning bounds.

11 Conclusion

Inference-time learning offers a route to robust adaptation and long-context processing, especially when paired with efficient sequence architectures such as SSMs and recurrent hybrids. It also introduces new risks because untrusted inputs can become training signals that modify model behavior. This paper proposes a practical guardrail loop based on write pressure monitoring, conservative pre-update gating, and canary drift rollback. The framework emphasizes defense in depth, observability, and auditability. A toy open-source implementation is provided to enable rapid experimentation and evaluation.

References

[1] Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei A. Efros, and Moritz Hardt. Test-time training with self-supervision for generalization under distribution shifts. *arXiv preprint arXiv:1909.13231*, 2019. URL https://arxiv.org/abs/1909.13231.

[2] Dequan Wang, Evan Shelhamer, Shaoteng Liu, Bruno Olshausen, and Trevor Darrell. Tent: Fully test-time adaptation by entropy minimization. *arXiv preprint arXiv:2006.10726*, 2020. URL https://arxiv.org/abs/2006.10726.

[3] OWASP Foundation. Owasp top 10 for large language model applications. https://owasp.org/www-project-top-10-for-large-language-model-applications/, 2025.

[4] Jingwei Yi, Yueqi Xie, Bin Benjamin Zhu, Emre Kiciman, Guangzhong Sun, Xing Xie, and Fangzhao Wu. Benchmarking and defending against indirect prompt injection attacks on large language models. *arXiv preprint arXiv:2312.14197*, 2023. URL https://arxiv.org/abs/2312.14197.

[5] Andy Zou, Zifan Wang, Nicholas Carlini, Milad Nasr, J. Zico Kolter, and Matt Fredrikson. Universal and transferable adversarial attacks on aligned language models. *arXiv preprint arXiv:2307.15043*, 2023. URL https://arxiv.org/abs/2307.15043.

[6] Xiaogeng Liu, Zhiyuan Yu, Yizhe Zhang, Ning Zhang, and Chaowei Xiao. Automatic and universal prompt injection attacks against large language models. *arXiv preprint arXiv:2403.04957*, 2024. URL https://arxiv.org/abs/2403.04957.

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017. URL https://arxiv.org/abs/1706.03762.

[8] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *arXiv preprint arXiv:2205.14135*, 2022. URL https://arxiv.org/abs/2205.14135.

[9] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023. URL https://arxiv.org/abs/2312.00752.

[10] Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. Retentive network: A successor to transformer for large language models. *arXiv preprint arXiv:2307.08621*, 2023. URL https://arxiv.org/abs/2307.08621.

[11] Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Huanqi Cao, Xin Cheng, Michael Chung, Matteo Grella, Kranthi Kiran GV, Xuzheng He, Haowen Hou, Przemyslaw Kazienko,

Jan Kocon, Jiaming Kong, Bartlomiej Koptyra, Hayden Lau, Krishna Sri Ipsit Mantri, Ferdinand Mom, Atsushi Saito, Xiangru Tang, Bolun Wang, Johan S. Wind, Stanislaw Wozniak, Ruichong Zhang, Zhenyuan Zhang, Qihang Zhao, Peng Zhou, Jian Zhu, and Rui-Jie Zhu. Rwkv: Reinventing rnns for the transformer era. *arXiv preprint arXiv:2305.13048*, 2023. URL https://arxiv.org/abs/2305.13048.

[12] Opher Lieber, Barak Lenz, Hofit Bata, Gal Cohen, Jhonathan Osin, Itay Dalmedigos, Erez Safahi, Shaked Meirom, Yonatan Belinkov, Shai Shalev-Shwartz, Omri Abend, Raz Alon, Tomer Asida, Amir Bergman, Roman Glozman, Michael Gokhman, Avashalom Manevich, Nir Ratner, Noam Rozen, Erez Shwartz, Mor Zusman, and Yoav Shoham. Jamba: A hybrid transformer-mamba language model. *arXiv preprint arXiv:2403.19887*, 2024. URL https://arxiv.org/abs/2403.19887.

[13] Tomer Asida et al. Jamba-1.5: Hybrid transformer-mamba models at scale. *arXiv preprint arXiv:2408.12570*, 2024. URL https://arxiv.org/abs/2408.12570.

[14] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021. URL https://arxiv.org/abs/2106.09685.

[15] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized large language models. *arXiv preprint arXiv:2305.14314*, 2023. URL https://arxiv.org/abs/2305.14314.

[16] Matan Ben-Tov, Mor Geva, and Mahmood Sharif. Universal jailbreak suffixes are strong attention hijackers. *arXiv preprint arXiv:2506.12880*, 2025. URL https://arxiv.org/abs/2506.12880.

[17] Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines. In *Proceedings of the 29th International Conference on Machine Learning (ICML)*, 2012.

[18] Jacob Steinhardt, Pang Wei Koh, and Percy Liang. Certified defenses for data poisoning attacks. *arXiv preprint arXiv:1706.03691*, 2017. URL https://arxiv.org/abs/1706.03691.

[19] Yubin Qu, Song Huang, Tongtong Bai, Xingya Wang, and Yongming Yao. Badcodeprompt: Backdoor attacks against prompt engineering of large language models for code generation. *Automated Software Engineering*, 32(17), 2025. doi: 10.1007/s10515-024-00485-2. URL https://doi.org/10.1007/s10515-024-00485-2.

[20] David Montgomery. Ttt input gradient monitor: A toy sandbox for write-pressure monitoring and inference-time learning guardrails. https://github.com/DMontgomery40/ttt_eval, 2026.

[21] Jimmy Ba, Geoffrey E. Hinton, Volodymyr Mnih, Joel Z. Leibo, and Catalin Ionescu. Using fast weights to attend to the recent past. *arXiv preprint arXiv:1610.06258*, 2016. URL https://arxiv.org/abs/1610.06258.

[22] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400*, 2017. URL https://arxiv.org/abs/1703.03400.

[23] Sewon Min, Mike Lewis, Luke Zettlemoyer, and Hannaneh Hajishirzi. Metaicl: Learning to learn in context. *arXiv preprint arXiv:2110.15943*, 2021. URL https://arxiv.org/abs/2110.15943.

[24] Yanda Chen, Ruiqi Zhong, Sheng Zha, George Karypis, and He He. Meta-learning via language model in-context tuning. *arXiv preprint arXiv:2110.07814*, 2021. URL https://arxiv.org/abs/2110.07814.

[25] Julian Coda-Forno, Marcel Binz, Zeynep Akata, Matthew Botvinick, Jane X. Wang, and Eric Schulz. Meta-in-context learning in large language models. *arXiv preprint arXiv:2305.12907*, 2023. URL https://arxiv.org/abs/2305.12907.

[26] Jiongxiao Wang, Fangzhou Wu, Wendi Li, Jinsheng Pan, G. Edward Suh, Z. Morley Mao, Muhao Chen, and Chaowei Xiao. Fath: Authentication-based test-time defense against indirect prompt injection attacks. *arXiv preprint arXiv:2410.21492*, 2024. URL https://arxiv.org/abs/2410.21492.

[27] Yulin Chen, Haoran Li, Yuexin Li, Yue Liu, Yangqiu Song, and Bryan Hooi. Topicattack: An indirect prompt injection attack via topic transition. *arXiv preprint arXiv:2507.13686*, 2025. URL https://arxiv.org/abs/2507.13686.

[28] Zongze Li, Jiawei Guo, and Haipeng Cai. System prompt poisoning: Persistent attacks on large language models beyond user injection. *arXiv preprint arXiv:2505.06493*, 2025. URL https://arxiv.org/abs/2505.06493.

[29] Advik Raj Basani and Xiao Zhang. Gasp: Efficient black-box generation of adversarial suffixes for jailbreaking llms. *arXiv preprint arXiv:2411.14133*, 2024. URL https://arxiv.org/abs/2411.14133.

[30] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *arXiv preprint arXiv:1612.00796*, 2016. URL https://arxiv.org/abs/1612.00796.