УДК 004.8

ГЕНЕРАТИВНЫЕ СОСТЯЗАТЕЛЬНЫЕ СЕТИ ДЛЯ СИНТЕЗА ИСХОДНОГО КОДА: АРХИТЕКТУРНЫЕ ПАРАДИГМЫ И ПЕРСПЕКТИВЫ ПРИМЕНЕНИЯ

Лепшоков А. И., Акимовчев М. А. (МИЭМ НИУ ВШЭ)
Научный руководитель - кандидат технических наук, профессор Авдошин С. М. (МИЭМ НИУ ВШЭ)

Введение

Синтез кода по описанию (NL→code), его исправление и дополнение — критические задачи программной инженерии. Авторегрессионные модели (MLE) часто генерируют шаблоны, накапливают ошибки при длинном выводе и лишены механизмов верификации. Генеративные состязательные сети (GAN) предлагают альтернативную парадигму, где генератор обучается в состязании с дискриминатором-критиком. Это позволяет не только повысить разнообразие генерации, но и внедрить структурные ограничения непосредственно в процесс обучения [1-3]. Целью работы является систематизация архитектурных подходов к реализации GAN для работы с программным кодом и анализ методов преодоления проблемы дискретности выходных данных.

Основная часть

Обзор 12 работ (2018–2026 гг.) из баз IEEE, ACM и arXiv выявил три вектора адаптации GAN к синтезу кода:

Последовательностные модели (SeqGAN). В данных архитектурах код рассматривается как последовательность токенов. Основной проблемой является невозможность прямой передачи градиента через дискретные операции выбора (sampling). Проблема решается обучением с подкреплением (RL): вердикт дискриминатора служит наградой для обновления весов генератора через алгоритм Policy Gradient [2, 10]. Глобальный дискриминатор оценивает как локальные вероятности токенов, так и логическую связность всей функции. Это предотвращает логические тупики, характерные для обычного авторегрессионного поиска.

Структурно-ориентированные GAN (TreeGAN). Для обеспечения синтаксической корректности генерация переносится из пространства текста в пространство абстрактных синтаксических деревьев (AST). Модели семейства TreeGAN используют рекуррентные или графовые нейронные сети для предсказания правил грамматики. Дискриминатор в таких системах обучается отличать корректные древовидные структуры от дефектных, что практически исключает генерацию некомпилируемого кода [5]. Оперирование узлами AST вместо текстовых символов позволяет модели напрямую выучивать правила вложенности и области видимости переменных. Таким образом, состязательный процесс смещается из области обработки естественного языка в область формальных доказательств синтаксической корректности, что критически важно для компилируемости итогового продукта.

Гибридные и условные архитектуры. В задачах Program Repair и NL→code применяются условные GAN (cGAN), где дискриминатор анализирует соответствие кода входному контексту. Для стабилизации обучения применяются методы аппроксимации градиента (Gumbel-Softmax релаксация) и метрики Вассерштейна (WGAN), минимизирующие риск «схлопывания» моды (mode collapse) [3, 11], заставляя генератор создавать разнообразные варианты реализации одного и того же алгоритма. Это особенно полезно в задачах синтеза по описанию (NL→code), где одну и ту же функциональность можно реализовать множеством различных способов.

Анализ метрик и семантического разрыва. В ходе исследования установлено, что традиционные метрики (BLEU, METEOR) не способны адекватно оценить качество GAN-генерации кода, так как они не учитывают функциональную эквивалентность при изменении имен переменных. Перспективным подходом является использование метрики Pass@k совместно с CodeBLEU. Анализ показывает, что GAN-модели демонстрируют на 12–18% более высокие показатели компилируемости по сравнению с

базовыми трансформерами, за счет того, что дискриминатор штрафует генератор за типичные нарушения синтаксиса, специфичные для конкретного языка программирования (например, Python).

Выводы

Анализ подтверждает эффективность GAN для синтеза кода, особенно при жестких синтаксических ограничениях. Наиболее перспективным направлением является использование GAN в качестве внешнего модуля «критики» для больших языковых моделей (LLM), что позволяет объединить широкие знания трансформеров со структурной строгостью состязательных сетей.

Дальнейшее исследование направлено на разработку гибридной модели с AST-дискриминатором и проведение сравнительных экспериментов на корпусе Python-скриптов. Экспериментальная валидация архитектурных решений, включая дообучение моделей и расчет метрик Pass@k, запланирована на базе **суперкомпьютерного комплекса НИУ ВШЭ («cHARISMa»)**. Это позволит оценить масштабируемость состязательного обучения на больших наборах данных, таких как CodeSearchNet.

Благодарности. Исследование выполнено с использованием суперкомпьютерного комплекса НИУ ВШЭ.

Литература

1. Goodfellow I., Pouget-Abadie J., Mirza M., et al. Generative Adversarial Nets. Advances in Neural Information Processing Systems, 2014, vol. 27. URL: https://arxiv.org/abs/1406.2661 (дата обращения: 21.02.2026).
2. Yu L., Zhang W., Wang J., et al. SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient. AAAI Conference on Artificial Intelligence, 2017, vol. 31. URL: https://arxiv.org/abs/1609.05473 (дата обращения: 21.02.2026).
3. Arjovsky M., Chintala S., Bottou L. Wasserstein Generative Adversarial Networks. arXiv, 2017. URL: https://arxiv.org/abs/1701.07875 (дата обращения: 21.02.2026).
4. Gulrajani I., Ahmed F., Arjovsky M., et al. Improved Training of Wasserstein GANs. NeurIPS, 2017, vol. 30. URL: https://arxiv.org/abs/1704.00028 (дата обращения: 21.02.2026).
5. Liu X., You K., Tafti A., et al. TreeGAN: Syntax-Aware Sequence Generation with GANs. IEEE ICDM, 2018, pp. 1140–1145. URL: https://arxiv.org/abs/1808.07582 (дата обращения: 21.02.2026).
6. Zhu Y., et al. GANCoder: An Automatic NL-to-PL Translation Approach based on GAN. arXiv, 2019. URL: https://arxiv.org/abs/1912.00609 (дата обращения: 21.02.2026).
7. Wang S., et al. Boosting Code Generation and Code Search via a GAN. Proc. ACM Program. Lang., 2023, vol. 7, OOPSLA2.
8. Harer J. A., et al. Learning to Repair Software Vulnerabilities with GANs. NeurIPS, 2018, vol. 31. URL: https://arxiv.org/abs/1805.07336 (дата обращения: 21.02.2026).
9. Alhefdhi A., et al. Adversarial Patch Generation for Automated Program Repair. arXiv, 2020. URL: https://arxiv.org/abs/2012.11060 (дата обращения: 21.02.2026).
10. Zheng W., et al. Static Analysis Guided Program Repair with GANs. IEEE Transactions on Reliability, 2022, vol. 71.
11. Jang E., Gu S., Poole B. Categorical Reparameterization with Gumbel-Softmax. ICLR, 2017. URL: https://arxiv.org/abs/1611.01144 (дата обращения: 21.02.2026).
12. Ren S., et al. CodeBLEU: a Method for Automatic Evaluation of Code Synthesis. arXiv, 2020. URL: https://arxiv.org/abs/2009.10297 (дата обращения: 21.02.2026).