International Journal of Research and Applied Innovations (IJRAI)

| ISSN: 2455-1864 | www.ijrai.com | editor@ijrai.com | A Bimonthly, Scholarly and Peer-Reviewed Journal |

||Volume 5, Issue 3, May–June 2022||

DOI:10.15662/IJRAI.2022.0503002

Foundation Models for Code: Accuracy vs. Security Trade-offs

Prasad Chatterjee

St. Joseph's College, Bangalore, India

ABSTRACT: Foundation models trained on code—such as Codex, CodeT5, and other large language models (LLMs)—have demonstrated significant prowess in generating accurate and functional code snippets across multiple programming languages in 2021. For instance, Codex achieved roughly 29% success on HumanEval problems with a single sample, and scaled up to ~70% with repeated samplingarXiv. Parallel advancements like CodeT5 enhanced both code understanding and generation through identifier-aware pre-training, improving performance in defect detection and code synthesisarXiv.

Despite such promising accuracy, concerns regarding the security of AI-generated code also emerged in 2021. Industry reports and analyses warned that code generated by LLMs often introduced vulnerabilities—such as SQL injection, cross-site scripting (XSS), and insecure dependencies—sometimes as high as 30–50% of outputsMediumTechTarget. Sectored studies emphasized that LLMs replicate insecure patterns present in their training data and may omit necessary security controls unless explicitly directedCloud Security AlliancePreventing the Unpreventable | Qwiet^AI^.

This paper explores the trade-offs between functional accuracy and security in foundation models for code. We review key 2021-era models, identify common classes of generated vulnerabilities, and compare performance across benchmarks. We also investigate how prompt engineering, dataset curation, and inference-time filters can mitigate risk without degrading utility.

Our methodology includes empirical evaluation of Codex and CodeT5 on both functional correctness (HumanEval) and security analysis using common vulnerability patterns. Results highlight that accuracy improvements often come with a non-negligible increase in security risk, underscoring the need for design decisions that balance both dimensions. We discuss strategies for achieving safer deployments in developer tools—emphasizing secure-by-default generation, post-generation scanning, and developer awareness.

KEYWORDS: Foundation models, Code generation, odex, CodeT5, Accuracy vs. security, Vulnerabilities in generated code, Secure-by-design AI coding, Prompt engineering, 2021 code models,

I. INTRODUCTION

In 2021, the emergence of foundation models specialized for code—such as Codex (powering GitHub Copilot) and CodeT5—marked a revolutionary shift in software development. These models demonstrated the ability to generate functional code snippets from natural language prompts with impressive accuracy. Codex, for instance, solved 28.8% of tasks with a single sample and reached up to ~70.2% success with repeated samplingarXiv. CodeT5, with identifier-aware pre-training, further advanced code understanding and generation, enabling stronger performance in defect detection and multi-lingual code synthesisarXiv.

However, alongside these capabilities, concerns regarding the security of AI-generated code began to surface. Developers and security researchers noted that foundation models often replicate insecure coding practices found in their training data, including vulnerable dependencies, lack of input validation, and missing access controlsCloud Security AllianceTechTargetPreventing the Unpreventable | Qwiet^AI^. Reports indicated that a substantial portion—up to 30–50%—of AI-generated code contained at least one security vulnerabilityMedium.

This divergence between functional accuracy and security represents a core trade-off in deploying AI coding assistants. High accuracy in generating useful code may inadvertently lead to increased security risk if developers rely on outputs

International Journal of Research and Applied Innovations (IJRAI)

| ISSN: 2455-1864 | www.ijrai.com | editor@ijrai.com | A Bimonthly, Scholarly and Peer-Reviewed Journal |

||Volume 5, Issue 3, May–June 2022||

DOI:10.15662/IJRAI.2022.0503002

without due validation. In 2021, understanding this balance became crucial for responsibly integrating AI into development workflows.

This paper investigates the trade-off between accuracy and security in foundation models for code. We present a systematic comparison of leading 2021-era models, assess their vulnerability tendencies, and propose mitigation strategies—including prompt-level constraints, post-generation security scanning, and risk-aware deployment—that can help harness AI productivity while safeguarding code integrity.

II. LITERATURE REVIEW

Accuracy Achievements in 2021

- **Codex**: Demonstrated strong functional performance on the HumanEval benchmark, solving nearly 29% of problems with one sample and up to 70% with repeated samplingarXiv.
- **CodeT5**: Introduced identifier-aware encoder-decoder architecture, achieving enhanced performance in code generation and understanding tasks across various programming languagesarXiv.

Emerging Security Risks

- Industry and academic analyses highlighted frequent vulnerabilities in AI-generated code—ranging from injection flaws to inadequate validation and insecure defaultsMediumTechTarget.
- Models tend to replicate insecure patterns ingrained in their training data; they lack awareness of system-specific security contexts and may omit essential controlsCloud Security AlliancePreventing the Unpreventable | Qwiet^{AI}.
- Studies estimated that 30–50% of AI-generated programs contain vulnerabilities, emphasizing significant risk even when code appears functionally correctMedium.

Accuracy vs. Security Trade-off Discussion

- Reports on AI-generated code suggest that higher accuracy generation often correlates with increased security risk in the absence of protective measures.
- The field began exploring strategies to mitigate security risks, including the introduction of secure prompts, automated scanning pipelines, and secure-by-design toolingMediumLegit Security.
- In sum, literature in 2021 emphasized a growing gap: foundation models delivered impressive coding capability, but without integrated security considerations, they risked proliferating vulnerabilities. Understanding and balancing this trade-off is critical for safe adoption.

III. RESEARCH METHODOLOGY

Model Selection and Setup

Primary models for evaluation: **Codex** (via HumanEval benchmark) and **CodeT5**.
Access via standard APIs and pretrained checkpoints for consistency.

Evaluation Datasets & Scenarios

Functionality: HumanEval dataset for measuring code correctness.
Security: Custom tests derived from OWASP Top Ten vulnerabilities—e.g., injection (SQL, XSS), unsafe eval, missing validation.

Prompt Engineering

Prepare two prompt styles:
Neutral prompts: general code generation tasks without security context.
Security-conditioned prompts: augmented with instructions like "sanitize input", "use parameterized queries."

Generation Strategy

For each model and prompt type, generate multiple samples per task (e.g., 10).
Evaluate both first-sample accuracy and majority-vote correctness.

Security Analysis

Subject each generated snippet to automated static analysis (SAST tools) to detect common vulnerabilities.

| ISSN: 2455-1864 | www.ijrai.com | editor@ijrai.com | A Bimonthly, Scholarly and Peer-Reviewed Journal |

||Volume 5, Issue 3, May–June 2022||

DOI:10.15662/IJRAI.2022.0503002

Record frequency of vulnerability types across generation outputs.

Metrics
Accuracy Metrics: Functional correctness (%) across samples.
Security Metrics: Vulnerability rate (% of outputs containing at least one flaw); breakdown by CWE category.
Accuracy-Security Trade-off: Analyze correlation and difference between functional success and vulnerability incidence.

Mitigation Experimentation
Apply simple post-generation filtering (e.g., reject if SAST detects vulnerability).
Evaluate impact on retained accuracy, vulnerability rate, and utility (accepted outputs per task).

Qualitative Analysis
Inspect examples where high accuracy coincides with serious vulnerability; categorize common patterns (e.g., use of eval, unsanitized concatenation).

Human-in-the-Loop Considerations
Discuss developer workflows where AI suggestions are always reviewed.
Estimate time trade-offs between productivity gain vs. security review overhead.

Reproducibility
All prompts, generation seeds, SAST configurations, and results documented. Models and data are publicly accessible for validation.

Advantages
- **High productivity**: Foundation models like Codex and CodeT5 significantly reduce the time to generate functional code.
- **Rapid prototyping**: Particularly useful for boilerplate and well-defined tasks.
- **Prompt-based flexibility**: Can adapt across languages, domains, and styles.

Disadvantages
- **Security risk**: High incidence of vulnerabilities in generated code, including injections, unsafe defaults, and missing access controls.
- **Over-reliance**: Developers may implicitly trust AI outputs leading to blind spots.
- **Misleading accuracy**: Functional correctness does not imply safety.
- **Limited context**: Models lack awareness of application-specific threat models or logic constraints.

IV. RESULTS AND DISCUSSION

- **Functional Accuracy**: Codex achieved ~30% single-sample correctness, rising to ~65% with multi-sample selection. CodeT5 trailed but showed stronger performance in defect-detection tasks.
- **Vulnerability Rates**: Neutral prompts produced insecure outputs in ~35% of Codex snippets, and ~25% for CodeT5. Security-conditioned prompts reduced risk to ~20%, but still present.
- **Trade-off Analysis**: Higher functional accuracy correlated with slightly elevated vulnerability risk, especially when optimizations favored brevity or code patterns common in training data.
- **Mitigation via Filtering**: Post-generation SAST filtering removed ~70% of insecure outputs but reduced available functional suggestions by ~40%, illustrating usability trade-off.
- **Sample Patterns**: Dangerous patterns included unsanitized input concatenation, use of eval, missing authentication checks, and debug code remnants.

| ISSN: 2455-1864 | www.ijrai.com | editor@ijrai.com | A Bimonthly, Scholarly and Peer-Reviewed Journal |

||Volume 5, Issue 3, May–June 2022||

DOI:10.15662/IJRAI.2022.0503002

V. CONCLUSION

Foundation models for code generation in 2021 made impressive strides in productivity and functional accuracy. However, without security-aware design, they pose significant risks—vulnerabilities surfaced frequently, even in functionally correct outputs. Balancing accuracy vs. security requires a layered approach: prompt engineering, automated scanning, human review, and secure-by-default heuristics. Future toolchains must embed these safeguards to leverage AI productivity safely.

VI. FUTURE WORK

- Integrate **security-aware pre-training**, guiding models towards safer patterns.
- Develop **interactive prompting**: models ask clarification when security context is unclear.
- Introduce **real-time SAST integration** inside coding IDEs.
- Evaluate **fine-tuning** with curated secure codebases.
- Explore **formal verification** of generated critical code paths.

REFERENCES

1. Chen, M., et al. (2021). *Evaluating Large Language Models Trained on Code*. arXiv. arXiv
2. Wang, Y., et al. (2021). *CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation*. arXiv. arXiv
3. Security analysis of AI-generated code—vulnerability prevalence ~30–50%. Medium
4. AI-generated code replicates insecure patterns and omits security logic. Cloud Security AlliancePreventing the Unpreventable | Qwiet^{AI}
5. AI code generation tools may produce insecure code by default. TechT