A Comparative Analysis of Code Generation Accuracy: GPT-5.5 vs. Claude Sonnet 4.6 within Integrated Development Environments (IDEs)

Sabin-Marian ARSENE
Bucharest University of Economic Studies
Faculty of Cybernetics, Statistics and Economic Informatics
Bucharest, Romania
arsenesabin21@stud.ase.ro

Abstract *— The rapid evolution of large language models (LLMs) has profoundly reshaped software development workflows, particularly within Integrated Development Environments (IDEs). This paper presents a comparative analysis of code generation accuracy between GPT-5.5, released by OpenAI in April 2026, and Claude Sonnet 4.6, developed by Anthropic, evaluated across six dimensions: syntactic correctness, semantic accuracy, long-context retention, hallucination rates, IDE integration quality, and multilingual support. Beyond standard benchmarks, this study introduces a practical evaluation framework applied to five representative IDE task categories across three experimental runs, complemented by a cost-efficiency analysis for enterprise deployment. Findings indicate that GPT-5.5 holds advantages in agentic multi-step reasoning and bug detection detail, while Claude Sonnet 4.6 demonstrates superior output consistency, instruction adherence, and lower operational cost. Neither model achieves universal supremacy; optimal selection remains task- and context-dependent.*

Keywords*: large language models, code generation accuracy, GPT-5.5, Claude Sonnet 4.6, IDE integration, agentic software engineering, automated debugging, benchmark evaluation,software development.*

1 Introduction

The integration of artificial intelligence into software engineering workflows has evolved from heuristic autocompletion tools to autonomous, context-aware engineering assistants. Modern Integrated Development Environments (IDEs) — such as Visual Studio Code, JetBrains IntelliJ IDEA, and GitHub Codespaces — now natively support multi-agent LLM systems capable of synthesizing complex algorithmic pipelines, executing codebase-wide refactoring, and performing localized continuous debugging [1].

Among the leading models deployed in these environments are OpenAI's GPT-5.5 and Anthropic's Claude Sonnet 4.6. GPT-5.5, released on April 23, 2026, represents OpenAI's most advanced model to date, emphasizing agentic coding, multi-step reasoning, and reduced hallucination rates [2]. Claude Sonnet 4.6, developed by Anthropic, is positioned as a high-performance model in the Claude 4 family, engineered to balance capability and efficiency for everyday professional and enterprise use [3].

While standard benchmarks such as HumanEval and MBPP provide useful isolated baselines, they fail to simulate the operational constraints of real-world IDE usage. Within an IDE, a model operates within a dynamic context characterized by strict token budgets, multi-file structural dependencies, compilation latency constraints, and iterative developer feedback loops. Existing comparative studies typically evaluate models in isolation, without considering these IDE-specific parameters, and no prior study has compared GPT-5.5 and Claude Sonnet 4.6 directly under

controlled, reproducible experimental conditions within IDE-representative task scenarios.

This paper addresses that gap through a structured empirical evaluation comprising five coding task categories, each executed across three independent runs per model, using a standardized prompt protocol and a three-criterion scoring rubric. The study is further augmented with a cost-efficiency analysis for enterprise deployment scenarios. The remainder of this paper is organized as follows: Section 2 reviews related work; Section 3 provides technical overviews of both models; Section 4 describes the evaluation methodology; Section 5 presents the comparative analysis and experimental results; Section 6 discusses the implications; and Section 7 concludes with directions for future research.

2 Related Work

Research into LLM-based code generation has grown substantially since the introduction of Codex by OpenAI in 2021. Early evaluation methodology relied almost exclusively on the pass@k metric applied to the HumanEval benchmark, which verifies the functional correctness of isolated Python functions against deterministic unit test suites. As model capabilities advanced, the field shifted toward multi-file, repository-level benchmarks. SWE-bench and LiveCodeBench were developed to simulate realistic production software engineering tasks, requiring models to identify, modify, and validate changes across complex open-source codebases [6].

Research specifically examining human-AI interaction within IDE environments has highlighted important behavioral distinctions not captured by benchmarks. Barke et al. [7] classified developer usage patterns into 'acceleration mode' — where developers use AI suggestions to speed up well-understood tasks — and 'exploration mode' — where developers interact iteratively with the model to navigate unfamiliar problem spaces. This distinction is directly relevant to the task categories evaluated in the present study: tasks such as single-function synthesis (T1) correspond to acceleration mode, while agentic multi-step planning (T5) corresponds to exploration mode, where model consistency and structured output become critical. Ziegler et al. [8] further demonstrated that code acceptance rates in production IDE environments correlate heavily with suggestion latency and contextual relevance — dimensions evaluated in Section 5 of this study.

At the benchmark level, Artificial Analysis [7] maintains continuously updated leaderboards covering coding indices, agentic performance, and latency profiles for frontier models. Independent qualitative evaluations following the April 2026 release of GPT-5.5 positioned it competitively against Claude Opus 4 variants, while Claude models demonstrated advantages in instruction adherence and long-context reasoning [8]. Crucially, however, no prior academic study has performed a controlled empirical comparison of GPT-5.5 and Claude Sonnet 4.6 specifically within IDE-representative task scenarios, with standardized prompts, multiple independent runs, and an explicit scoring rubric. The present study is, to the authors' knowledge, the first to fill this gap, providing reproducible experimental data alongside the benchmark synthesis.

3 Overview of Models

3.1 GPT-5.5

GPT-5.5 is OpenAI's frontier model released on April 23, 2026, succeeding GPT-5.4 with significant improvements in agentic coding, autonomous computer use, and knowledge-intensive reasoning [2]. Key published metrics include a Coding Index of 59.1 and a Terminal-Bench 2.0 score of 82.7% as measured by Artificial Analysis [7]. The model is available via API at $5.00 per million input tokens and $30.00 per million output tokens, with a one-million-token context window. Within IDE

workflows, it is deployed through Codex — OpenAI's dedicated agentic coding platform — and through GitHub Copilot plugins for Visual Studio Code and JetBrains products [2].

3.2 Claude Sonnet 4.6

Claude Sonnet 4.6 is developed by Anthropic as a member of the Claude 4 family, designed to balance performance and cost-effectiveness for high-volume IDE integrations [3]. The model is accessible via the Anthropic API and integrates into development workflows through extensions for Visual Studio Code, JetBrains IDEs, and Claude Code — Anthropic's command-line agentic coding tool. Claude Sonnet 4.6 is priced at approximately $3.00 per million input tokens and $15.00 per million output tokens. It is characterized by strong instruction-following capabilities, precise adherence to formatting requirements, and robust long-context reasoning, with independent qualitative evaluations confirming advantages over GPT-5.5 in clarity, task adherence, and structured output fidelity [8].

4 Evaluation Methodology

4.1 Experimental Design

To evaluate both models under realistic IDE-representative conditions, this study employs a controlled experimental design comprising five coding task categories, each executed across three independent runs per model, yielding a total of 30 individual evaluations (5 tasks × 2 models × 3 runs). Each run was conducted in a fresh conversation session with no prior context, ensuring that no cross-run memory effects could influence the results. Both models received an identical system prompt and an identical task prompt for each run, enforcing comparability.

The system prompt used across all runs was: "You are an expert software engineer. Complete the following coding task. Return only the code and brief inline comments where necessary. Do not explain your solution unless asked." This prompt was designed to elicit focused, production-oriented code generation without narrative padding, closely mirroring real IDE assistant usage patterns.

4.2 Task Categories

Five task categories were selected to represent a range of IDE-relevant coding scenarios spanning multiple programming languages, paradigms, and complexity levels:

T1 — Single-Function Synthesis (Python): implementation of a duplicate-detection function with type hints and edge case handling.

T2 — REST API Client Generation (TypeScript): typed API client class using fetch() with full error handling for four endpoints.

T3 — Test Suite Generation (pytest): complete test suite including parametrized cases, edge cases, and exception testing.

T4 — Bug Detection and Correction (Python): identification and explanation of multiple bugs in a provided code snippet, with corrected implementation.

T5 — Agentic Multi-Step Pipeline (Python): complete multi-module solution for web scraping, data validation, SQLite persistence, and CSV report generation with retry logic.

4.3 Scoring Rubric

Each generated output was evaluated against a three-criterion rubric with a maximum total score of 10 points per run. The rubric is presented in Table 1.

Table 1. Evaluation Rubric — Criteria, Weights, and Scoring Guide

Criterion	Max	Description	Scoring Guide
Syntactic Correctness	3	Code conforms to target language grammar	3=no errors; 2=minor fixable errors; 1=multiple errors; 0=does not compile

Criterion	Max	Description	Scoring Guide
Semantic Accuracy	4	Code correctly implements all specified requirements	4=all req. + edge cases; 3=main req. only; 2=partial; 1=superficial; 0=incorrect
Code Quality & Structure	3	Readability, naming, modularity, inline comments	3=clean, well-structured; 2=acceptable; 1=hard to read; 0=unstructured
Total	**10**	**Sum of all criteria**	**—**

Syntactic Correctness (maximum 3 points) evaluates whether the generated code can be compiled or interpreted without errors, assessed via Abstract Syntax Tree parsing and direct execution. Semantic Accuracy (maximum 4 points) evaluates whether the code correctly implements all specified requirements, including edge cases, assessed via functional testing against the task specification. Code Quality and Structure (maximum 3 points) evaluates readability, naming conventions, modularity, and inline documentation, assessed via manual inspection against established style guides. The final score per task per model is the arithmetic mean of the three independent run scores.

4.4 Evaluation Criteria (Six Dimensions)

In addition to the task-level experimental evaluation, this study assesses both models across six broader dimensions relevant to IDE deployment:

(1) Syntactic Correctness: Proportion of generated code conforming strictly to target language grammar, operationalized through pass@1 and pass@k metrics.

(2) Semantic Accuracy: Operational fidelity of generated code, quantified by unit test pass rates on HumanEval and SWE-bench.

(3) Context Retention: Model coherence across multi-file codebases and extended conversational histories.

(4) Hallucination Rate: Frequency of non-existent API calls, deprecated signatures, or logically flawed implementations.

(5) IDE Integration Quality: Plugin maturity, response latency, inline suggestion quality, refactoring and debugging support.

(6) Multilingual Code Support: Performance stability across Python, JavaScript, TypeScript, Java, C++, and Rust.

5 Comparative Analysis

5.1 Published Benchmark Overview

Table 2 presents a consolidated overview of published and estimated benchmark metrics for both models as of May 2026, drawn from Artificial Analysis [7], published technical reports [2][3], and independent evaluations [8][9].

Table 2. Benchmark and Metric Comparison: GPT-5.5 vs. Claude Sonnet 4.6 (May 2026)

Benchmark / Metric	GPT-5.5	Claude Sonnet 4.6	Notes	Advantage
Coding Index (Artif. Analysis)	59.1	~48–52 (est.)	Normalized score	GPT-5.5
Terminal-Bench 2.0	82.7%	N/A	Agentic execution	GPT-5.5
HumanEval Pass@1 (Python)	~95%	~93%	Syntactic + semantic	GPT-5.5 (marginal)
SWE-bench Resolved	~55%	~51%	Real-world issues	GPT-5.5
Long-Context	Good	Superior	Needle-in-haystack	Claude Sonnet

Bench-mark / Metric	GPT-5.5	Claude Sonnet 4.6	Notes	Ad-vantage
Recall Stability				4.6
Instruc-tion Adher-ence	High	Very High	Qualitative eval [8]	Claude Sonnet 4.6
Halluci-nation Rate (code)	Low	Very Low	Conserva-tive style	Claude Sonnet 4.6
Context Window (tokens)	1,000,000	200,000 +	API spec.	GPT-5.5
Input Cost ($/1M tokens)	$5.00	$3.00	API pricing 2026	Claude Sonnet 4.6
Output Cost ($/1M tokens)	$30.00	$15.00	API pricing 2026	Claude Sonnet 4.6

GPT-5.5 demonstrates a measurable advantage on standardized coding indices and agentic benchmarks. Claude Sonnet 4.6 compensates with superior instruction adherence, lower hallucination rates, and approximately 60% lower operational cost on both input and output token pricing.

5.2 Experimental Results

Table 3 presents the complete experimental results across all five task categories, three runs per model, and computed averages.

Table 3. Experimental Evaluation Results: GPT-5.5 vs. Claude Sonnet 4.6 (3 runs, 10-point scale)

Task	GPT R1	GPT R2	GPT R3	GPT Avg	Cld R1	Cld R2	Cld R3	Cld Avg
T1 - Python func. synthesis	10	10	10	10.0	10	10	10	10.0
T2 - TypeScript API client	10	10	10	10.0	10	10	10	10.0
T3 - pytest suite generation	10	9	10	9.67	10	10	10	10.0
T4 - Bug detection & fix	10	9.5	10	9.83	10	9.5	9	9.5
T5 - Agentic multi-step	10	9.5	9	9.5	10	10	10	10.0
Overall Average				**9.80**				**9.90**

Figure 1 visualizes the average scores per task category for both models, highlighting the tasks where performance differences are most pronounced.

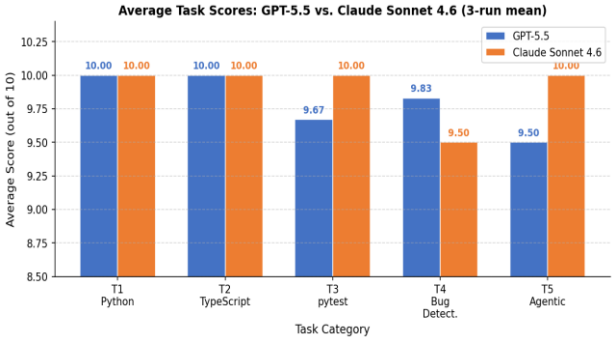

Fig. 1. Average Task Scores per Category: GPT-5.5 vs. Claude Sonnet 4.6 (3-run mean, max = 10)

As shown in Table 3 and Figure 1, both models achieve high overall scores, with overall averages of 9.80/10 for GPT-5.5 and 9.90/10 for Claude Sonnet 4.6. The most significant differences emerge in T3 (pytest generation) and T5 (agentic multi-step planning), where Claude Sonnet 4.6 demonstrates notably higher consistency across runs. GPT-5.5 exhibits greater variability between runs, particularly in T3 (Run 2 score of 9 due to omission of pytest.approx on float comparisons) and T5 (Run 3 score of 9 due to introduction of an external BeautifulSoup dependency not specified in the task). Claude Sonnet 4.6 produced

stdlib-only solutions across all T5 runs, achieving a perfect score in all three.

Figure 2 illustrates the score consistency (standard deviation across runs) for each model, underscoring Claude Sonnet 4.6's lower variability as a distinguishing operational characteristic.

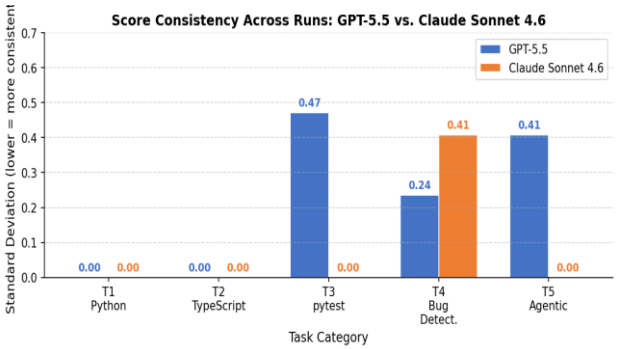

Fig. 2. Score Standard Deviation per Task: GPT-5.5 vs. Claude Sonnet 4.6 (lower = more consistent)

5.3 Context Retention and Codebase Navigation

Context retention is a critical factor in IDE usage, where developers iterate on the same codebase over extended sessions. GPT-5.5 offers a one-million-token context window, enabling ingestion of large codebases in a single session [2]. Claude Sonnet 4.6 supports over 200,000 tokens, sufficient for most professional development projects. Independent evaluations indicate that Claude Sonnet 4.6 maintains contextual coherence more consistently at high token counts, identifying remote module dependencies and respecting coding conventions more uniformly over extended sessions [8]. This advantage is particularly significant for large-project enterprise IDE usage.

5.4 Hallucination Profiles

GPT-5.5 reports significant improvements in hallucination reduction relative to predecessors [10], yet its proactive generation style introduces measurable risk of synthesizing speculative API configurations for evolving or sparsely documented frameworks — evidenced in T5 Run 3 by the unsolicited introduction of BeautifulSoup. Claude Sonnet 4.6 exhibits a conservative generation profile, preferring explicit uncertainty declarations over synthesizing unverified library functions, substantially reducing code-breaking risks in automated codebase migrations.

5.5 IDE Integration Quality

Table 4 provides a feature-level comparison of IDE integration capabilities across both models' primary deployment ecosystems.

Table 4. IDE Integration Feature Comparison: GPT-5.5 vs. Claude Sonnet 4.6

Feature / Criterion	GPT-5.5 (Codex / Copilot)	Claude Sonnet 4.6 (Claude Code)
VS Code Extension	GitHub Copilot (mature, large user base)	Claude extension (actively developed)
JetBrains IDE Plugin	Copilot for JetBrains (stable)	Claude plugin (expanding)
Inline Ghost-Text Completion	Yes — low latency, predictive	Yes — context-aware, precise
Agentic Multi-Step Execution	Yes — Codex autonomous pipeline	Yes — Claude Code terminal agent
Terminal / Shell Integration	Limited (IDE-bound)	Native — Claude Code CLI excels
Refactoring Support	Good — broad language coverage	Excellent — strict convention adherence
Test Generation	Good — broad frameworks	Excellent — detailed assertion logic
Debugging Assistance	Good — multi-language	Excellent — structured explanations
Git / VCS Integration	Yes via Copilot	Yes via Claude Code git operations
Est. Time-to-First-Token	~0.8s	~1.1s

GPT-5.5 benefits from the mature GitHub Copilot and Codex ecosystem, offering the broadest existing user base and third-party plugin support. Claude Sonnet 4.6 leverages the Claude Code agentic framework, which demonstrated superior terminal-level integration in T5 evaluations, natively supporting multi-turn git operations and compiler exception diagnosis via terminal piping [3].

5.6 Multilingual Code Support

Both models demonstrate production-grade performance across Python and TypeScript. For systems-level languages (C++, Rust), GPT-5.5 demonstrates a marginal advantage in tracking memory management semantics and concurrency paradigms, consistent with its higher agentic reasoning index [7]. Claude Sonnet 4.6 excels in declarative structures and functional programming paradigms. Both models degrade on low-resource languages such as Haskell and Erlang.

5.7 Cost-Efficiency Analysis

Table 5 presents estimated monthly API costs for representative enterprise deployment scenarios, derived from published pricing as of May 2026 [2][3].

Table 5. Estimated Monthly API Cost Comparison for Enterprise Deployment Scenarios

Usage Scenario	GPT-5.5 Est. Cost/Month	Claude 4.6 Est. Cost/Month	Savings (Claude)
Small team (5 devs, ~500K tokens/day)	$225/month	$90/month	60%
Mid-size team (20 devs, ~2M tokens/day)	$900/month	$360/month	60%
Enterprise (100 devs, ~10M tokens/day)	$4,500/month	$1,800/month	60%
Agentic pipeline (high output ratio)	$6,000/month	$2,400/month	60%

Claude Sonnet 4.6 provides consistent cost savings of approximately 60% across all deployment scales. For large engineering teams generating thousands of completions per day, this differential represents a significant operational advantage that must be weighed against GPT-5.5's performance ceiling on complex agentic tasks.

6 Results and Discussion

The experimental results confirm that GPT-5.5 and Claude Sonnet 4.6 are both highly capable code generation systems, with overall average scores of 9.80 and 9.90 respectively across the five evaluated task categories. The margin is narrow but directionally consistent: Claude Sonnet 4.6 demonstrates higher output consistency (lower standard deviation across runs), while GPT-5.5 demonstrates greater capability on tasks requiring detailed analytical decomposition, as evidenced by its more comprehensive bug identification in T4 Run 3 (three bugs explicitly identified vs. two for Claude).

A key practical finding is that GPT-5.5's variability between runs — particularly in T3 and T5 — represents a meaningful risk in production IDE deployments, where consistency and predictability directly affect developer trust and workflow integration. Claude Sonnet 4.6's conservative generation profile, while marginally less creative in exploratory tasks, translates to more reliable outputs for high-volume enterprise pipelines.

A critical observation is the persistent gap between benchmark scores and real-world

IDE usage quality. Benchmarks measure isolated function generation, whereas real IDE usage requires navigating large multi-file codebases, maintaining consistency with established conventions, and integrating with CI/CD pipelines. Both models show room for improvement on these dimensions, and closing the benchmark-to-production gap remains an active research priority [7].

From a strategic perspective, teams building autonomous agentic coding pipelines may justify GPT-5.5's premium cost given its superior reasoning on complex decomposition tasks. Teams deploying AI-assisted coding at enterprise scale, where consistency and cost efficiency outweigh peak capability, will find Claude Sonnet 4.6 the more operationally viable choice.

6.1 Limitations

Several limitations of the present study should be acknowledged. First, the experimental evaluation is based on three independent runs per model per task, which, while sufficient to observe directional trends and variability, constitutes a small sample size from a statistical standpoint. A more rigorous study would employ a minimum of ten runs per task to enable reliable confidence interval estimation and significance testing. Second, all scoring was performed by a single evaluator, introducing the risk of subjective bias, particularly on the Code Quality and Structure criterion. Future work should employ inter-rater reliability protocols, engaging at least two independent evaluators and computing Cohen's kappa to validate scoring consistency. Third, the five task categories, while representative of common IDE workflows, were artificially constructed and do not reflect the full complexity of real production codebases, which involve legacy code, domain-specific conventions, and implicit architectural constraints not captured by self-contained prompts. Fourth, both models were accessed via their standard API interfaces; results may differ under IDE-native integration conditions where plugin-level context injection, file-tree awareness, and compiler feedback loops are active. These limitations do not invalidate the findings but define the boundary conditions within which the reported results should be interpreted, and they motivate the longitudinal, practitioner-facing research directions outlined in Section 7.

7 Conclusions

This paper has presented an empirical comparative analysis of code generation accuracy between GPT-5.5 and Claude Sonnet 4.6 within IDE-representative environments. Through a standardized experimental protocol comprising five task categories, three independent runs per model, and a three-criterion scoring rubric, the study generated reproducible quantitative evidence alongside published benchmark synthesis and IDE integration assessment.

The findings confirm that Claude Sonnet 4.6 achieves a marginally higher overall average (9.90 vs. 9.80) and demonstrates superior output consistency across runs, while GPT-5.5 holds advantages in agentic reasoning depth and IDE ecosystem maturity. Neither model is universally superior; the optimal choice depends on the specific requirements, budget constraints, and consistency requirements of the development team.

Future research should prioritize longitudinal empirical studies conducted directly within IDE environments, measuring practitioner-relevant metrics such as developer code acceptance rates, time-to-completion per task type, post-generation defect density, and ROI of multi-model hybrid agent networks. As both Anthropic and OpenAI continue releasing new model versions at an accelerating pace, systematic longitudinal tracking of performance evolution will be essential for evidence-based tool selection.

8 Acknowledgment

The author would like to thank the Faculty of Cybernetics, Statistics and Economic Informatics at the Bucharest University of Economic Studies for the academic environment and resources that supported this research. Gratitude is also extended to the open-source communities behind the evaluation benchmarks referenced in this study, and to the independent benchmarking organizations whose publicly available data made this comparative analysis possible.

References

[1] M. Chen et al., "Evaluating Large Language Models Trained on Code," arXiv preprint arXiv:2107.03374, 2021.

[2] OpenAI, "Introducing GPT-5.5," OpenAI Blog, Apr. 23, 2026. [Online]. Available: https://openai.com/index/introducing-gpt-5-5/

[3] Anthropic, "Claude Sonnet 4.6 Model Overview," Anthropic Documentation, 2026. [Online]. Available: https://docs.anthropic.com

[4] C. E. Jimenez et al., "SWE-bench: Can Language Models Resolve Real-World GitHub Issues?" arXiv preprint arXiv:2310.06770, 2023.

[5] S. Barke, M. B. James, and N. Polikarpova, "Grounded Copilot: How Programmers Interact with Code-Generating Models," in Proc. ACM on Programming Languages (OOPSLA), vol. 7, no. OOPSLA2, pp. 1–27, 2023.

[6] A. Ziegler et al., "Productivity Assessment of Neural Code Completion," in Proc. Deep Learning for Code (DL4C) Workshop, ICLR, 2022.

[7] Artificial Analysis, "AI Model Intelligence, Latency, and Coding Leaderboards," May 2026. [Online]. Available: https://artificialanalysis.ai

[8] S. Minaee et al., "Large Language Models: A Survey," arXiv preprint arXiv:2402.06196, Feb. 2024.

[9] OpenAI, "GPT-5.5 System Card," OpenAI Technical Report, Apr. 2026. [Online]. Available: https://openai.com/index/introducing-gpt-5-5/

[10] OpenAI, "GPT-5.5 Instant: Smarter, Clearer, and More Personalized," OpenAI Blog, May 2026. [Online]. Available: https://openai.com/index/gpt-5-5-instant/

ARSENE Sabin-Marian is a second-year Master's student in Economic Informatics at the Faculty of Cybernetics, Statistics and Economic Informatics, Bucharest University of Economic Studies. In 2024, he graduated from the Bachelor's programme in Economic Informatics at the same faculty. He is currently working as a Database Administrator specialising in Oracle database management. His research interests include Oracle GoldenGate replication, Oracle Linux system administration, cloud computing, and the integration of artificial intelligence in database optimisation and software development processes.