

Zen4-Coder-Pro: Advanced Agentic Software Engineering at Full Lifecycle Scale

Technical Whitepaper v2026.02

Zen LM Research Team

research@zenlm.org

papers.zenlm.org

February 2026

Abstract

Zen4-Coder-Pro extends the Zen4-Coder family to 72 billion parameters with capabilities spanning the complete software engineering lifecycle: architecture design, implementation, testing, documentation, security review, and deployment. Built on the Mixture of Experts (MoE) architecture with an expanded expert pool and a reinforced agentic training protocol, Zen4-Coder-Pro achieves 96.8% on HumanEval, 67.3% on SWE-bench, 71.2% on SWE-bench Verified, and 91.4% on LiveCodeBench. Beyond benchmark performance, Zen4-Coder-Pro demonstrates end-to-end project development capability: given a specification, it can design system architecture, implement all components, write comprehensive tests, generate documentation, and produce CI/CD pipeline configurations. This paper describes the model's architecture, training protocol, capability profile, and deployment considerations.

Contents

1	Introduction	3
1.1	Key Capabilities	3
1.2	Model Overview	3
2	Architecture	3
2.1	Expanded MoE Architecture	3
2.2	Extended Context Architecture	4
2.3	Architecture Reasoning Module	4
3	Training Methodology	4
3.1	Training Data	4
3.2	Long-Horizon Agentic Fine-Tuning	5
3.3	Constitutional Engineering Principles	5
4	Evaluation	6
4.1	Standard Code Generation Benchmarks	6
4.2	SWE-bench Performance	6
4.3	End-to-End Project Development	6
4.4	Security Review Quality	7
4.5	Architecture Design Quality	7

5	Agentic System Integration	7
5.1	Tool Ecosystem	7
5.2	Multi-Agent Coordination	7
6	CI/CD Integration	8
6.1	Native Pipeline Support	8
6.2	Pipeline Validation	8
7	Deployment	8
7.1	Inference Requirements	8
7.2	Latency Profile	8
8	Safety and Ethics	9
8.1	Autonomous Action Guardrails	9
8.2	License Compliance	9
9	Related Work	9
10	Conclusion	9

1 Introduction

The history of software engineering automation has progressed through distinct phases: from static analysis tools, to test generation frameworks, to LLM-based code completion, and now to agentic systems capable of autonomous multi-step engineering work. Each transition required not just larger models but qualitatively different training objectives and capability profiles.

Zen4-Coder-Pro represents the current frontier of this progression. At 72B parameters with 18B active per token, it surpasses the capability ceiling of 32B models on tasks requiring deep long-horizon planning, sophisticated architectural reasoning, and cross-cutting concerns such as security, performance, and maintainability.

1.1 Key Capabilities

The distinctive capabilities of Zen4-Coder-Pro relative to Zen4-Coder (32B) are:

1. **Architecture Design:** Given a requirements document, Zen4-Coder-Pro produces complete system architecture specifications including component diagrams, API contracts, data models, and technology selection rationale.
2. **End-to-End Project Development:** From specification to deployable artifact, including implementation, test suites, documentation, and infrastructure-as-code.
3. **Security Review:** Identify vulnerabilities at the design and implementation level, categorized by OWASP/CVSS severity with remediation guidance.
4. **CI/CD Integration:** Generate and validate CI/CD pipeline configurations (GitHub Actions, GitLab CI, Tekton) that correctly build, test, and deploy the generated code.
5. **Performance Analysis:** Profile execution bottlenecks, reason about asymptotic complexity, and propose data-structure or algorithmic improvements.

1.2 Model Overview

Table 1: Zen4-Coder-Pro Model Specification

Parameter	Value
Architecture	Mixture of Experts (MoE)
Total Parameters	72B
Active Parameters per Token	18B
Number of Experts	128
Top- k Active Experts	4
Context Window	256K tokens
Supported Languages	92 programming languages
Version	v2026.02
Release Date	February 2026

2 Architecture

2.1 Expanded MoE Architecture

Zen4-Coder-Pro uses an expanded instance of the MoE architecture with 128 experts (compared to 64 in Zen4-Coder). The expanded expert pool provides finer-grained specialization: whereas Zen4-Coder has experts that specialize broadly by language family (systems, scripting,

functional), Zen4-Coder-Pro develops sub-experts that specialize by both language and task type.

The router in Zen4-Coder-Pro is a two-level hierarchical router. The first level selects a domain:

$$d^* = \arg \max_d \sigma(W_d \cdot h_t), \quad d \in \{\text{design, impl, test, sec, ops}\} \quad (1)$$

The second level selects experts within the chosen domain partition:

$$\alpha_i = \frac{\exp(W_{d^*,i} \cdot h_t)}{\sum_{j \in \mathcal{E}_{d^*}} \exp(W_{d^*,j} \cdot h_t)}, \quad i \in \text{top}_k(\mathcal{E}_{d^*}) \quad (2)$$

This hierarchical structure reduces routing collisions and improves expert utilization, with measured expert utilization entropy of 0.91 (maximum possible: 1.0) compared to 0.76 for flat routing at the same parameter count.

2.2 Extended Context Architecture

Zen4-Coder-Pro supports a 256K token context window through a combination of:

1. **Sliding Window Attention:** Local attention for most layers with a window size of 8K tokens.
2. **Global Attention Layers:** Every 8th transformer layer uses full global attention over the entire context.
3. **Positional Extrapolation:** YaRN-based position encoding that extrapolates beyond training context lengths to enable dynamic extension to 512K tokens when needed.

2.3 Architecture Reasoning Module

Zen4-Coder-Pro includes a specialized Architecture Reasoning Module (ARM) that operates at the design-level abstraction above individual code files. The ARM maintains a structured representation of:

- **Component graph:** Nodes are services/modules, edges are dependencies with annotated protocols.
- **Data flow:** How data structures transform as they flow through the system.
- **Invariant set:** System-level invariants (consistency requirements, capacity constraints, SLAs).
- **Decision log:** Architecture decisions made during the session with rationale.

The ARM enables coherent multi-session project development where architectural decisions made early constrain later implementation choices consistently.

3 Training Methodology

3.1 Training Data

Zen4-Coder-Pro was trained on a 9.2 trillion token corpus, extending Zen4-Coder’s data with additional long-horizon engineering artifacts:

Table 2: Pre-Training Data Composition

Source	Tokens (B)	Fraction
Public code repositories	4,100	44.6%
Architecture documents & RFCs	800	8.7%
Pull request & code review history	1,100	12.0%
Issue tracker & bug report corpora	700	7.6%
Security advisory databases	400	4.3%
Performance analysis reports	350	3.8%
CI/CD configs & DevOps scripts	400	4.3%
Documentation and API references	900	9.8%
Academic CS papers	350	3.8%
General natural language (filtered)	100	1.1%
Total	9,200	100%

3.2 Long-Horizon Agentic Fine-Tuning

The key training innovation in Zen4-Coder-Pro is Long-Horizon Agentic Fine-Tuning (LHAFT). Standard instruction fine-tuning optimizes single-turn or short-session performance. LHAFT constructs multi-session training trajectories that span the full development lifecycle of a project:

1. **Trajectory synthesis:** A curriculum of 50K synthetic project specifications was generated, each paired with a complete development trajectory (architecture design $\rightarrow$ implementation $\rightarrow$ tests $\rightarrow$ docs $\rightarrow$ CI/CD).
2. **Expert annotation:** A subset of 8K trajectories was reviewed and annotated by senior software engineers to provide quality signal on architecture choices, code quality, and test coverage.
3. **Reward shaping:** Trajectories are scored by a composite reward: code correctness (RLCE), test coverage, documentation completeness, CI pipeline validity, and security scan results.

$$r_{\text{LHAFT}} = w_1 r_{\text{correct}} + w_2 r_{\text{coverage}} + w_3 r_{\text{docs}} + w_4 r_{\text{ci}} + w_5 r_{\text{sec}} \quad (3)$$

with weights $w = [0.4, 0.2, 0.15, 0.15, 0.1]$ reflecting the relative importance of functional correctness.

3.3 Constitutional Engineering Principles

Zen4-Coder-Pro is trained with a set of constitutional software engineering principles that guide its outputs:

- Prefer explicit over implicit; surface assumptions in interfaces.
- Fail fast with precise error messages; never swallow failures silently.
- Minimize public API surface; keep interfaces small and orthogonal.
- Prove patterns before abstracting; duplicate a little before generalizing.
- Default to UTF-8, deterministic behavior, and reproducible builds.
- Never store credentials in plaintext; use environment variables or secret managers.

These principles are encoded as preference pairs in a Constitutional AI training stage, causing Zen4-Coder-Pro to internalize them as defaults rather than requiring explicit instruction.

4 Evaluation

4.1 Standard Code Generation Benchmarks

Table 3: Code Generation Benchmark Results

Model	HumanEval	MBPP+	HumanEval+	LiveCodeBench
Zen4-Coder-Pro (72B)	96.8%	91.7%	94.9%	91.4%
Zen4-Coder (32B)	95.2%	89.3%	93.1%	88.7%
Comparable 70B baseline	94.3%	88.6%	92.4%	87.3%

4.2 SWE-bench Performance

SWE-bench [1] evaluates the ability to resolve real GitHub issues with actual code patches applied to live repositories. Zen4-Coder-Pro achieves the best results in its parameter class on both the standard and Verified subsets.

Table 4: SWE-bench Results

Model	SWE-bench	SWE-bench Verified
Zen4-Coder-Pro (72B)	67.3%	71.2%
Zen4-Coder (32B)	58.4%	61.2%
Comparable 70B baseline A	61.8%	65.4%
Comparable 70B baseline B	59.3%	63.1%
Comparable 70B baseline C	63.7%	67.9%

4.3 End-to-End Project Development

We introduce the End-to-End Project Benchmark (E2EPB), a new evaluation suite measuring the quality of complete project artifacts produced by a single agentic run from a specification document. E2EPB covers 50 projects across web backend, CLI tooling, data pipeline, and embedded systems domains.

Each project is evaluated across six dimensions:

Table 5: E2EPB Results (score 0–100 per dimension)

Dimension	Zen4-Coder-Pro	Zen4-Coder	Human Baseline
Architecture coherence	84.3	71.2	91.7
Implementation correctness	88.6	79.4	94.2
Test coverage	82.1	70.8	86.4
Documentation completeness	91.4	78.3	83.1
CI/CD pipeline validity	87.7	68.4	89.3
Security posture	79.3	62.1	88.7
Composite score	85.6	71.7	88.9

Zen4-Coder-Pro achieves 96.3% of the human baseline composite score, a substantial improvement over Zen4-Coder’s 80.6%.

4.4 Security Review Quality

On the OWASP Security Review Benchmark (250 code samples with known vulnerability categories):

Table 6: Security Review Benchmark Results

Model	Detection Rate	False Positive Rate	Severity Accuracy	Fix Quality
Zen4-Coder-Pro (72B)	91.2%	4.3%	87.6%	83.4%
Zen4-Coder (32B)	81.7%	6.8%	79.3%	74.2%
Static analyzer baseline	74.3%	12.1%	71.4%	N/A

4.5 Architecture Design Quality

On the Architecture Design Evaluation (ADE) benchmark, expert reviewers (senior engineers with 10+ years of experience) rated architecture documents produced by each model on a 5-point scale:

Table 7: Architecture Design Evaluation (1–5 MOS)

Model	Coherence	Completeness	Feasibility	Overall
Zen4-Coder-Pro (72B)	4.1	4.3	4.0	4.1
Zen4-Coder (32B)	3.4	3.7	3.5	3.5
Human senior engineer	4.6	4.4	4.5	4.5

5 Agentic System Integration

5.1 Tool Ecosystem

Zen4-Coder-Pro is designed for deep integration into agentic pipelines. Beyond the tool categories supported by Zen4-Coder, Zen4-Coder-Pro adds:

- **Architecture diagram generation:** Produce Mermaid/PlantUML diagrams from component graph representations.
- **Dependency vulnerability scanning:** Query CVE databases and package vulnerability APIs during dependency selection.
- **Performance profiling:** Analyze flamegraphs and profiling output to identify bottlenecks.
- **Documentation generation:** Synthesize API documentation, user guides, and architecture decision records.
- **Deployment validation:** Validate Kubernetes manifests, Terraform configs, and Dockerfile correctness.

5.2 Multi-Agent Coordination

In large-scale engineering workflows, Zen4-Coder-Pro serves as an orchestrator agent coordinating a team of specialized sub-agents (Zen4-Coder-Flash instances for rapid exploration, domain-specific tools for linting and static analysis). The orchestrator decomposes project tasks, assigns subtasks to sub-agents, integrates results, and resolves conflicts.

This multi-agent architecture achieves 23% higher E2EPB composite scores compared to a single Zen4-Coder-Pro instance when projects exceed 10K lines of target code, by parallelizing independent module development while maintaining global architectural coherence through the ARM.

6 CI/CD Integration

6.1 Native Pipeline Support

Zen4-Coder-Pro includes native understanding of major CI/CD platforms: GitHub Actions, GitLab CI, Jenkins, CircleCI, Tekton, and ArgoCD. When generating project artifacts, it automatically produces pipeline configurations that:

1. Build the project in a clean environment.
2. Run the generated test suite with coverage reporting.
3. Execute security scanning (SAST, dependency audit).
4. Build and push container images with attestation.
5. Deploy to staging and validate with smoke tests.
6. Gate production deployment behind manual approval or automated quality thresholds.

6.2 Pipeline Validation

Generated pipelines are validated by a symbolic pipeline interpreter that checks for common errors (missing environment variables, incorrect dependency ordering, unreachable jobs) before returning them to the user. In evaluation on 200 project generation tasks, 94.1% of generated pipelines executed without modification, compared to 71.3% for Zen4-Coder.

7 Deployment

7.1 Inference Requirements

Table 8: Inference Resource Requirements

Configuration	Hardware	Throughput
FP8 (recommended)	4 × H100 80GB	310 tok/s
BF16	8 × H100 80GB	240 tok/s
INT4	2 × H100 80GB	420 tok/s

7.2 Latency Profile

Table 9: Latency Benchmarks (FP8, 4×H100)

Task	P50 Latency	P95 Latency
Function completion (256 tok)	0.8s	1.4s
File-level generation (2K tok)	6.2s	9.8s
Full module (10K tok)	31.4s	48.7s
Architecture document (5K tok)	16.1s	24.3s

8 Safety and Ethics

8.1 Autonomous Action Guardrails

Given Zen4-Coder-Pro’s increased agentic capabilities, additional safety guardrails are applied:

- **Destructive action confirmation:** Any file deletion, database migration, or infrastructure teardown requires explicit confirmation before execution.
- **Secret handling:** Generated code never stores credentials in plaintext; the model enforces secret manager patterns (Vault, AWS Secrets Manager, environment variables).
- **Scope limiting:** Agentic sessions operate within declared repository and permission boundaries; out-of-scope actions are blocked and logged.
- **Audit trail:** All tool calls and file mutations in agentic sessions are logged with timestamps and rationale for post-hoc review.

8.2 License Compliance

Zen4-Coder-Pro tracks the licenses of code it references or adapts. When generating code that incorporates patterns from copyleft-licensed sources, the model identifies the license implications and suggests alternatives or proper attribution.

9 Related Work

Full software lifecycle automation has been approached through agentic LLM systems [2, 3], multi-agent coordination frameworks [4, 5], and specialized planning architectures [6]. Zen4-Coder-Pro unifies these capabilities within a single model trained end-to-end rather than relying on prompt engineering over general-purpose models, enabling more coherent and reliable behavior across the full engineering lifecycle.

10 Conclusion

Zen4-Coder-Pro advances the state of the art in agentic software engineering, achieving exceptional benchmark performance on SWE-bench and LiveCodeBench while introducing new capabilities in architecture design, end-to-end project development, and CI/CD integration. The 72B MoE architecture with hierarchical expert routing provides the reasoning depth required for cross-cutting concerns such as security, performance, and long-horizon planning. Zen4-Coder-Pro represents a practical step toward AI systems that can serve as capable collaborators across the full software development lifecycle.

References

- [1] Jimenez, C. et al. (2024). SWE-bench: Can Language Models Resolve Real-World GitHub Issues? ICLR 2024.
- [2] Cognition AI. (2024). Devin: The First AI Software Engineer.
- [3] Yang, J. et al. (2024). SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. arXiv:2405.15793.
- [4] Qian, C. et al. (2023). Communicative Agents for Software Development. arXiv:2307.07924.

- [5] Hong, S. et al. (2023). MetaGPT: Meta Programming for Multi-Agent Collaborative Framework. arXiv:2308.00352.
- [6] Qiao, B. et al. (2023). TaskWeaver: A Code-First Agent Framework. arXiv:2311.17541.