Zen AI Model Family

Zen Coder

Agentic Code Generation Models (4B - 1T)

Technical Report v2.0

Zach Kelling*
research@hanzo.ai

Zoo Labs Foundation
foundation@zoo.ngo

December 2025

Abstract

We present **Zen Coder**, a family of agentic code generation models ranging from 4B to 1T parameters, trained on the **Zen Agentic Dataset**—8.47 billion tokens of real-world Claude Code sessions, git history, and professional software development spanning 15 years across 1,452 repositories. Unlike synthetic datasets, our training data captures actual debugging workflows, multi-file refactoring decisions, tool use patterns, and error recovery from production AI development. The model family includes dense models (4B, 24B, 123B) and MoE architectures (358B, 1T), covering edge deployment to frontier capabilities.

Contents

*zach@lux.network

1 Introduction

The emergence of agentic AI systems—where models interact with tools, execute multi-step plans, and maintain context across complex workflows—has created demand for models specifically trained on real agentic programming patterns. **Zen Coder** addresses this by training on actual Claude Code debug sessions rather than synthetic instruction-following data.

1.1 Model Family

Model	Size	Base	VRAM	Context	Status
Zen Coder 4B	4B	Zen-4B-Instruct	8 GB	32K	Trained
Zen Coder 24B	24B	Devstral Small 2	24 GB	256K	Trained
Zen Coder 123B	123B	Devstral 2	128 GB	256K	Training
Zen Coder Max	358B (MoE)	GLM-4.7	180 GB	200K	Planned
Zen Coder Ultra	1T (MoE)	Kimi K2	256 GB	128K	Planned

Table 1: Zen Coder Model Family

1.2 Key Innovations

- **Real Agentic Data**: Trained on actual Claude Code sessions, not synthetic instruction data
- **Production Code**: 15 years of battle-tested software across AI, Web3, cryptography
- **Multi-Architecture**: Dense (4B-123B) and MoE (358B-1T) models for different deployment needs
- **Open Training**: Full training framework available via zen-trainer

2 Training Data: Zen Agentic Dataset

The **Zen Agentic Dataset** comprises 8.47 billion tokens derived from real software development and Claude Code interactions, distinguishing it from synthetic datasets.

Data Source	Tokens (B)	%	Description
Git History	4.03	48%	15 years of commits, diffs, source files
Claude Debug Sessions	2.42	29%	Real debugging workflows with tool use
Claude Conversations	1.14	13%	Architecture discussions, code reviews
Claude Interactions	0.86	10%	Multi-turn coding assistance
Total	**8.47**	100%	3.35M training samples

Table 2: Zen Agentic Dataset Composition

2.1 Dataset Statistics

Metric	Value
Total Tokens	8.47 billion
Training Samples	3.35 million
Validation Samples	100,000
Total Size	27 GB
Repositories	1,452
Time Span	15 years (2010-2025)

Table 3: Dataset Statistics

2.2 Domain Coverage

2.2.1 Agentic AI & LLM Infrastructure

- Model Context Protocol (MCP) - 260+ tool implementations
- Multi-agent orchestration - Claude, GPT-4, Gemini, Zen integrations
- Agent frameworks - Planning, memory, tool use, reflection
- LLM Gateway - Unified proxy for 100+ providers

2.2.2 Web3 & Blockchain

- Smart contracts - Solidity, Vyper (ERC20, ERC721, ERC1155, DeFi)
- Consensus engines - Snow family, BFT, DAG-based protocols
- Cross-chain bridges - Multi-VM architecture
- DeFi protocols - AMMs, lending, staking, governance

2.2.3 Cryptography & Security

- Post-quantum cryptography - Kyber, Dilithium, SPHINCS+
- Threshold cryptography - MPC, secret sharing, DKG
- Zero-knowledge proofs - zkSNARKs, zkSTARKs experimentation
- Key management - HD wallets, hardware integration

2.2.4 Modern Development

- Full-stack TypeScript - Next.js 14+, React 18+, Node.js
- Systems programming - Rust, Go, Python, C/C++
- DevOps - Docker, Kubernetes, CI/CD pipelines
- Real-time systems - Event sourcing, CQRS, message queues

2.3 Language Distribution

Tier 1 (Core)	Tier 2 (Infrastructure)	Tier 3 (Specialized)
Python	SQL	Solidity
TypeScript	Bash/Shell	C/C++
JavaScript	YAML/TOML	Protobuf
Rust	Dockerfile	GraphQL
Go	Makefile	Move

Table 4: Language Coverage by Tier

3 Model Architectures

3.1 Dense Models (4B, 24B, 123B)

The smaller models use dense transformer architectures optimized for different deployment scenarios:

Model	Base Architecture	Layers	Hidden Dim
Zen Coder 4B	Zen-4B-Instruct	40	2560
Zen Coder 24B	Devstral Small 2	56	5120
Zen Coder 123B	Devstral 2	80	8192

Table 5: Dense Model Specifications

3.2 MoE Models (358B, 1T)

The larger models employ Mixture of Experts architectures for efficient scaling:

Model	Total Params	Active Params	Experts
Zen Coder Max	358B	60B	128 experts, top-8
Zen Coder Ultra	1T	150B	256 experts, top-8

Table 6: MoE Model Specifications

4 Training Methodology

4.1 Training Framework

We developed zen-trainer, an open-source framework supporting multiple backends:

- **MLX**: Apple Silicon optimization (M1/M2/M3 Pro/Max/Ultra)
- **Unsloth**: 2x faster NVIDIA training with memory optimization
- **DeepSpeed**: Multi-GPU and multi-node training

4.2 Fine-tuning Configuration

Model	LoRA r	LoRA α	Batch	LR
4B	64	128	4	2e-4
24B	32	64	2	1e-4
123B	16	32	1	5e-5
Max	16	32	1	5e-6
Ultra	8	16	1	1e-6

Table 7: QLoRA Fine-tuning Hyperparameters

4.3 Training Costs

For 3.35M samples (8.47B tokens) on 8xH200 @ $35/hr:

Model	Cloud Hours	Cloud Cost	Local (Mac Studio)
Zen Coder 4B	9h	$326	2 days (FREE)
Zen Coder 24B	23h	$814	5 days (FREE)
Zen Coder 123B	62h	$2,171	13 days (FREE)
Zen Coder Max	116h	$4,071	19 days (FREE)
Zen Coder Ultra	310h	$10,856	N/A (too large)

Table 8: Training Cost Estimates

5 Usage

5.1 Installation

```
pip install zen-trainer
```

Listing 1: Install zen-trainer

5.2 Training Example

```
from zen_trainer import ZenTrainer

trainer = ZenTrainer(
    model_key="zen-coder-4b",
    dataset_path="hanzoai/zen-agentic-dataset-private",
    output_dir="./output/zen-coder-4b",
)
trainer.train()
```

Listing 2: Fine-tuning with zen-trainer

5.3 Inference Example

```
from transformers import AutoModelForCausalLM, AutoTokenizer

model = AutoModelForCausalLM.from_pretrained(
    "zenlm/zen-coder-4b",
```

```
    torch_dtype="auto",
    device_map="auto"
)
tokenizer = AutoTokenizer.from_pretrained("zenlm/zen-coder-4b")

messages = [
    {"role": "user", "content": "Write a Python function to validate email addresses"}
]

text = tokenizer.apply_chat_template(messages, tokenize=False, add_generation_prompt=True)
inputs = tokenizer(text, return_tensors="pt").to(model.device)
outputs = model.generate(**inputs, max_new_tokens=512)
print(tokenizer.decode(outputs[0], skip_special_tokens=True))
```

Listing 3: Using Zen Coder for inference

6 What Makes This Unique

6.1 Real Agentic Programming

Unlike synthetic datasets, the Zen Agentic Dataset contains **actual Claude Code sessions** showing:

- Real debugging workflows with trial and error
- Complex multi-file refactoring decisions
- Architecture discussions and trade-offs
- Tool use patterns (file ops, search, git, tests)
- Error recovery and iterative refinement

6.2 Production Code Quality

- Code that shipped to production systems
- Security-audited smart contracts
- Performance-optimized infrastructure
- Battle-tested patterns from real deployments

7 Evaluation

Models are evaluated on agentic coding benchmarks including:

- **SWE-bench Verified**: Real GitHub issue resolution
- **TAU-Bench**: Tool-agent-user interaction
- **BFCL V3**: Berkeley Function Call Leaderboard
- **Terminal-Bench**: Terminal environment tasks
- **LiveCodeBench**: Real-time coding challenges

8 Licensing & Access

8.1 Models

- Zen Coder 4B, 24B: Apache 2.0
- Zen Coder 123B: Mistral Research License
- Zen Coder Max: GLM-4 License
- Zen Coder Ultra: MIT (Kimi K2 base)

8.2 Dataset Access

The Zen Agentic Dataset is available for research and commercial licensing. Contact `z@hanzo.ai` for access.

9 Supported Organizations

Organization	Focus	Role
Hanzo AI	AI infrastructure	Primary maintainer
Zen LM	Open model research	Model training
Zoo Labs	Decentralized AI	Research grants
Lux Network	AI compute settlement	Infrastructure

Table 9: Supporting Organizations

10 Conclusion

Zen Coder represents a new approach to code generation model training: using real agentic programming data rather than synthetic instructions. By training on actual Claude Code sessions from 15 years of production software development, these models learn genuine debugging patterns, tool use workflows, and the iterative nature of real programming.

The complete model family—from 4B for edge deployment to 1T for frontier capabilities—provides options for every use case. The open training framework (zen-trainer) enables the community to train custom models on their own data.

Links

- Models: huggingface.co/zenlm
- Dataset: huggingface.co/datasets/hanzoai/zen-agentic-dataset
- Training: github.com/zenlm/zen-trainer
- Website: zenlm.org
- Contact: z@hanzo.ai