ChatGPT

AI-Native Development Patterns

AI-native development treats machine-learning models as first-class components of software systems. Emerging tools like vLLM, Ray Serve, BentoML, Cursor 2.0 and autonomous testing frameworks enable developers to build, deploy and maintain AI applications more efficiently.

High-Throughput Model Serving

vLLM V1

- **Use case**: Serving large language models (LLMs) with high concurrency.
- **Pattern**: Deploy vLLM with fractionally allocated GPUs and persistent batch preparation to maximise throughput. vLLM's modular architecture and zero-overhead prefix caching deliver up to 1.7× speedup and lower memory usage [1].
- **Implementation**:

```
from vllm import LLM, SamplingParams
llm = LLM(model="gpt2", tensor_parallel_size=4)
outputs = llm.generate(["Hello", "How are you?"],
SamplingParams(max_tokens=32))
```

- **Benefits**: Higher QPS and lower cost per token.

Ray Serve

- **Use case**: Composing multiple models and business logic into scalable inference services.
- **Pattern**: Chain models using Ray Serve deployments with dynamic request batching and streaming responses [2].
- **Implementation**:

```
@serve.deployment
class Summarizer:
    async def __call__(self, request):
        ...
app = FastAPI()
app.route("/summarize")(Summarizer.bind())
```

- **Benefits**: Scales across nodes/GPUs; integrates with FastAPI for easy endpoints.

BentoML

- **Use case**: Packaging AI applications with multi-model pipelines and auto-generated APIs.
- **Pattern**: Create a Bento that bundles models, preprocessing code and dependencies, then deploy via BentoCloud or Kubernetes [3].
- **Implementation**:

```
import bentoml
@bentoml.service
class MyService:
    @bentoml.api
    async def generate(self, prompt: str) -> str:
        return model.generate(prompt)
```

- **Benefits**: Standardized deployment and adaptive batching across frameworks.

AI-Powered Development Workflows

Multi-Agent Coding with Cursor 2.0

- **Use case**: Parallelizing coding tasks (implementation, testing, documentation) among AI agents [4].
- **Pattern**: Define multiple agents with specific roles and assign them tasks using Cursor's multi-agent interface; they run concurrently using git worktrees.
- **Implementation**:
- Configure `agents.json` specifying goals for each agent (e.g., build feature, write tests).
- Use Cursor UI to run agents in parallel and review their pull requests.
- **Benefits**: Reduces human context switching and accelerates complex tasks.

GitHub Copilot Workspace

- **Use case**: Conversational planning and execution of development tasks within a repository [5].
- **Pattern**: Start a workspace session, describe your goal in natural language and let Copilot generate a step-by-step plan and code changes. Edit and run tests within the environment.
- **Benefits**: Encourages iterative design and collaboration with AI.

Autonomous Testing and Mutation Testing

- **AI-generated tests**: Use LLMs (e.g., GPT-4o) to generate large numbers of unit tests, boosting coverage [6].
- **Mutation testing**: Employ a tool like **Stryker Mutator** to introduce small mutations to code and ensure the test suite detects them [7].
- **Workflow**:
- Generate baseline tests with AI.
- Run mutation testing and inspect surviving mutants.
- Refine or augment tests to kill surviving mutants.
- **Benefits**: Ensures tests are robust and not just high-coverage but meaningful.

Responsible AI and Governance

- Integrate ethics and governance tools (Fiddler, Credo AI, IBM OpenScale, Arthur AI) into the machine-learning lifecycle to monitor bias, explain predictions and detect drift [8] [9] [10] [11].
- Align with emerging regulations (EU AI Act) by tracking compute used (>10^23 FLOPS), documenting training data and performing risk assessments [12].

Best Practices

- **Verify AI outputs**: Always review and test code generated by AI to avoid errors.
- **Monitor performance**: Use metrics and tracing to monitor throughput and latency when deploying models with vLLM, Ray Serve or BentoML.
- **Security first**: Secure model endpoints with mTLS and workload identities using SPIFFE/SPIRE [13].
- **Compliance**: Incorporate supply chain security (SLSA, in-toto) and AI governance frameworks early to avoid rework and regulatory penalties.

---

[1] vLLM V1: A Major Upgrade to vLLM's Core Architecture | vLLM Blog
https://blog.vllm.ai/2025/01/27/v1-alpha-release.html

[2] Ray Serve: Scalable and Programmable Serving — Ray 2.51.0
https://docs.ray.io/en/latest/serve/index.html

[3] bentoml · PyPI
https://pypi.org/project/bentoml/1.2.9/

[4] Introducing Cursor 2.0 and Composer · Cursor
https://cursor.com/blog/2-0

[5] GitHub Copilot Workspace: Welcome to the Copilot-native developer environment - The GitHub Blog
https://github.blog/news-insights/product-news/github-copilot-workspace/

[6] Generating 17,000 lines of working test code in less than an hour | early Blog
https://www.startearly.ai/post/openai-startearlyai-unit-tests-benchmark-generating-1000-unit-tests-under-an-hour

[7] Stryker Mutator
https://stryker-mutator.io/

[8] [9] [10] [11] Top 10 AI Model Governance Tools for Bias and Ethics Management (2025 Guide)
https://www.cloudnuro.ai/blog/top-10-ai-model-governance-tools-for-bias-and-ethics-management-2025-guide

[12] Overview of Guidelines for GPAI Models | EU Artificial Intelligence Act
https://artificialintelligenceact.eu/gpai-guidelines-overview/

[13] Goodbye Service API Keys: SPIFFE/SPIRE Workload Identity and Zero-Trust mTLS Across Kubernetes and Multi-Cloud in 2025 | DebuggAI
https://debugg.ai/resources/goodbye-service-api-keys-spiffe-spire-workload-identity-zero-trust-mtls-kubernetes-multi-cloud-2025