

TECHNICAL DOCUMENTATION

vibewithai

The Intelligent Developer SDK for Python

AI that performs engineering tasks — not a chat wrapper.

Version 0.1.1

Published 30 July 2026

Package pypi.org/project/vibewithai

Source github.com/ragulraj-d/vibewithai

Licence MIT

Contents

1	Overview
2	Installation
3	Quick Start
4	API Reference
5	Configuration
6	Architecture
7	Developer Guide
8	Best Practices
9	Troubleshooting
10	Frequently Asked Questions
11	Security Policy
12	Migration Guide
13	Changelog

Overview

```
import vibewithai

vibewithai.login("sk-proj-...", "AIza...", "sk-ant-...") # any keys, any order

result = vibewithai.check()

print(result.score)      # 87
print(result.summary)    # "3 issues found across 12 files (1 at or above high)."
```

You describe an **engineering task**. The SDK handles provider selection, context building, token budgeting, retries, and failover.

Why vibewithai

Most AI coding tools are a prompt box with a system message. This is a working SDK with an opinion about how AI should be used on real codebases.

Static analysis runs first	AST, Ruff, Bandit and Radon execute locally before any model call. Most questions are answered deterministically, for free. The model is reserved for reasoning that tools genuinely cannot do.
You never name a provider	Pass whatever API keys you have. Each is matched to its provider by key shape, verified with a live health check, and the first working one becomes active.
Failover is automatic	Rate limited, over quota, or a 5xx? The SDK switches to your next verified provider mid-request. A bad credential never triggers a switch — that would hide a real misconfiguration.
Structured results, not prose	Every command returns a <code>CommandResult</code> with a score, typed findings, metrics and token accounting. The CLI is just one renderer.
Nothing runs, nothing is overwritten	Generated code is returned as data and never executed. Files are never modified without explicit opt-in, and originals are backed up.
~2 ms import	Lazy attribute resolution means the SDK is usable inside pre-commit hooks and editor plugins. A test fails the build if import exceeds 100 ms.

Installation

```
pip install vibewithai                # core
pip install "vibewithai[analysis]"    # + Ruff, Bandit, Radon, LibCST
(recommended)
```

Requires Python 3.11+. Tested on 3.11–3.13 across Linux, macOS and Windows.

Runtime dependencies are deliberately minimal: `httpx`, `platformdirs`, `rich`, `typer`. No vendor SDKs — all 16 providers are reached over plain HTTP.

Quick start

1. Authenticate

```
import vibewithai

vibewithai.login(
    "sk-proj-xxxxxxx",    # OpenAI
    "AIzaSyxxxxxxx",     # Google Gemini
    "xai-xxxxxxx",       # xAI Grok
    "sk-ant-xxxxxxx",    # Anthropic
)
```

Or keep keys out of your code entirely:

```
export VIBEWITHAI_OPENAI_API_KEY=sk-proj-...
export ANTHROPIC_API_KEY=sk-ant-...           # conventional vendor variables work too
```

```
vibewithai.login()    # environment → stored credentials → local Ollama
```

Guided terminal setup:

```
vibewithai init
```

2. Run a task

```
result = vibewithai.review("src/app.py")

for finding in result.findings_at_or_above(vibewithai.Severity.HIGH):
    print(f"{finding.location} [{finding.severity}] {finding.title}")
    print(f"    {finding.detail}")
    print(f"    via {finding.source} (confidence {finding.confidence})")
```

`finding.source` tells you whether a result came from a deterministic tool

(`"ruff"`, `"bandit"`) or a model (`"ai:openai"`), so you always know how much to trust it.

3. Gate on it

```
result = vibewithai.security()

if not result:                # falsy only when the tool itself errored
    raise SystemExit(2)
if not result.passed:         # False when blocking findings exist
    raise SystemExit(result.exit_code)
```

The API

Nineteen commands, grouped by intent. Every one returns a `CommandResult`.

Understand

```
vibewithai.check()                # full quality assessment, scored 0–100
vibewithai.review("src/app.py")    # pull-request style review
vibewithai.explain("src/auth.py")  # what it does and why
vibewithai.ask("Why is login failing?")
vibewithai.chat("How should I structure this service?")
```

Improve

```
vibewithai.fix("src/app.py")       # propose corrections
vibewithai.refactor("src/app.py")  # restructure, behaviour preserved
vibewithai.optimize()              # find real performance problems
vibewithai.comment("src/app.py")   # docstrings where they are missing
```

Secure and ship

```
vibewithai.security()          # vulnerability audit with exploitability reasoning
vibewithai.deploy()           # production-readiness assessment
vibewithai.debug("KeyError: user_id", traceback=tb)
```

Create

```
vibewithai.test("src/app.py")   # generate pytest tests
vibewithai.document()           # API reference
vibewithai.readme()             # project README
vibewithai.generate("a FastAPI CRUD API for users")
vibewithai.coverage()           # measure via pytest --cov
```

Version control

```
vibewithai.commit()             # Conventional Commits message from staged diff
vibewithai.changelog(since="v1.2.0") # Keep a Changelog entry
```

Commands that produce files write **nothing** unless you pass `apply=True` :

```
result = vibewithai.fix("src/app.py")
print(result.artifacts["src/app.py"])      # review first
vibewithai.fix("src/app.py", apply=True)    # then apply; original is backed up
```

Command line

```
vibewithai          # welcome screen with detected environment
vibewithai init      # guided setup
vibewithai doctor    # diagnose the installation
vibewithai tutorial  # learn the commands in five minutes

vibewithai check     # exits 1 on findings at or above `high`
vibewithai review src/app.py
vibewithai security --fail-on medium
vibewithai fix src/app.py --yes

vibewithai check --format json      # machine-readable
vibewithai check --format html > report.html
vibewithai check --changed-only     # only files changed from HEAD
```

Exit codes: 0 clean · 1 findings at or above the threshold · 2 execution error.

Result objects

```
result = vibewithai.check()

result.score           # 0–100 quality score
result.status          # Status.WARNING
result.summary         # human-readable paragraph
result.findings        # list[Finding] – severity, location, source, confidence
result.suggestions     # list[Suggestion] – title, rationale, effort
result.metrics         # files, lines, complexity, cache hit rate
result.usage           # prompt/completion/cached token accounting
result.provider        # "openai"
result.execution_time  # seconds

result.passed          # quality gate
result.exit_code       # POSIX exit code
result.counts_by_severity # {'critical': 0, 'high': 1, ...}
result.findings_at_or_above(vibewithai.Severity.HIGH)

result.to_dict()       # JSON-serialisable, safe to log
result.to_json()
```

Render to any format:

```
from vibewithai.reports import render

open("report.html", "w").write(render(result, "html"))
open("report.md", "w").write(render(result, "markdown"))
```

Supported providers

All 16 work with an API key alone — no vendor SDKs, no extra dependencies.

CATEGORY	PROVIDERS
Frontier	OpenAI · Anthropic Claude · Google Gemini · xAI Grok
Aggregators	OpenRouter · Together AI · Fireworks AI · Hugging Face · GitHub Models
Direct	DeepSeek · Mistral AI · Cohere · Groq · Perplexity
Enterprise	Azure OpenAI
Local	Ollama — no key required

Provider selection, key detection and failover are entirely automatic. Adding a new OpenAI-compatible provider is a single entry in `providers/specs.py`.

Not yet supported: AWS Bedrock, Google Vertex AI and OCI Generative AI require cloud-native request signing (SigV4, service-account JWTs) rather than bearer tokens. Reach those models through OpenRouter or an OpenAI-compatible gateway in the meantime.

Configuration

Layered, highest precedence first: **call arguments** → **environment** → **project file**
→ **user file** → **defaults**.


```

default_provider = "openai"          # omit to use the first verified provider
offline          = false             # true = local static analysis only
max_context_tokens = 12000

[providers.openai]
model = "gpt-4o-mini"

[analysis]
analyzers      = ["ast", "ruff", "bandit", "radon"]
blocking_severity = "high"
exclude        = [".venv", "node_modules", "*_pb2.py"]

[auth]
fallback      = true                 # switch providers on rate limits and outages
preferred_order = ["anthropic", "openai"]

[cache]
enabled = true
ttl      = 86400

```

Credentials are never read from project files — only from environment

variables and the user-level credential store — so a key cannot be committed by accident. Full reference: [docs/configuration.md](https://docs.astral.sh/ruff/configuration.md).

Enterprise considerations

Security

- **Credentials never leak.** API keys are wrapped in a `Secret` type that masks itself in `str()`, `repr()`, f-strings, and refuses to pickle. Independently, a logging filter scrubs credential-shaped text from every record before emission.
- **Nothing is written without consent.** Commands that modify the workspace raise `ConfirmationRequiredError` unless explicitly authorised; originals are backed up; writes outside the project root are refused; every write is atomic.
- **AI output is never executed.** Not `eval`-ed, not `exec`-ed, not imported.
- **Owner-only credential storage** — `0600` file inside a `0700` directory, with a warning if permissions loosen.

- Full threat model: [SECURITY.md](#).

Data governance

Any AI command sends a **bounded, relevant slice** of your code — the file in question plus its direct collaborators — never the whole repository. For codebases that must not leave your network:

```
offline = true    # local static analysis only; zero network calls
```

Or run entirely on local models via Ollama.

CI/CD

```
- run: pip install "vibewithai[analysis]"
- run: vibewithai check --fail-on high --format json > report.json
env:
  VIBEWITHAI_OPENAI_API_KEY: ${ secrets.OPENAI_API_KEY }
```

Ready-made configurations for [GitHub Actions](#),

[GitLab CI](#), [Jenkins](#) and

[Azure DevOps](#).

Performance

Analysis results are cached against file content digests, so unchanged files are never re-analysed and never re-sent to a model. Independent files are analysed in parallel. Repeat runs on a large repository complete in milliseconds.

Extensibility

```
from vibewithai.plugins import register

register(provider=my_provider_spec)    # add a provider
register(analyzer=MyAnalyzer())       # add a static check
register(reporter=("sarif", to_sarif)) # add an output format
```

Plugins are also discovered automatically through the `vibewithai.plugins` entry-point group.

Documentation

GUIDE	
Installation	Platforms, extras, verification
Quick start	First run, walkthrough
Configuration	Every setting and environment variable
API reference	Complete public surface
Architecture	Layering and design decisions
Developer guide	Adding providers and analyzers
Best practices	Getting good results
Troubleshooting	Common failures and fixes
FAQ	

Project health

Tests	798 passing
Coverage	89%
Type checking	<code>mypy --strict</code> , clean
Linting	Ruff + Black, clean
Security scan	Bandit, clean
Import time	~2 ms
Runtime dependencies	4

Status

0.1.0 — first release. The full pipeline is implemented and tested end to end.

As a first release, the AI-backed commands have had far more automated testing than real-world mileage. Treat their output as a capable reviewer's opinion rather than ground truth — which is exactly why every finding carries a `confidence` score and a `source` . Deterministic static analysis is unaffected by this caveat.

Issues and feedback: github.com/ragulraj-d/vibewithai/issues

Contributing

Contributions are welcome — see [CONTRIBUTING.md](#) and the [Code of Conduct](#). To report a vulnerability, follow [SECURITY.md](#) rather than opening a public issue.

License

[MIT](#)

Installation

Requirements

- **Python 3.11+** (3.11, 3.12, 3.13 are tested in CI)
- Linux, macOS, or Windows

Install

```
pip install vibewithai
```

With local static-analysis backends (recommended — these run before any AI call and often answer the question without one):

```
pip install "vibewithai[analysis]"    # ruff, bandit, radon, libcst
```

Using [uv](#):

```
uv pip install "vibewithai[analysis]"  
uv tool install vibewithai            # CLI only, isolated
```

Verify

```
vibewithai --version  
vibewithai info
```

`info` prints your interpreter, platform and which optional backends are available. Its output excludes environment variables and home-directory paths, so it is safe to paste into a public bug report.

```
import vibewithai  
print(vibewithai.__version__)
```

Runtime dependencies

Deliberately small:

PACKAGE	WHY
<code>httpx</code>	One HTTP client for all providers, instead of per-vendor SDKs
<code>platformdirs</code>	Correct config/cache locations on every OS
<code>rich</code>	CLI rendering
<code>typer</code>	CLI argument parsing

None are imported by `import vibewithai` — see [architecture](#).

Development install

```
git clone https://github.com/ragulraj-d/vibewithai.git
cd vibewithai
python3.11 -m venv .venv && source .venv/bin/activate
pip install -e ".[dev]"
pre-commit install
pytest
```

Upgrading

```
pip install --upgrade vibewithai
```

Check [CHANGELOG.md](#) before upgrading across a minor version.

Uninstalling

```
pip uninstall vibewithai
```

Stored credentials and cache are left in place. To remove them too:

```
vibewithai config path      # shows the exact locations
vibewithai cache clear
```

Quick Start

1. Authenticate

Pass API keys positionally, in any order. The provider is detected from the key itself — you never name it:

```
import vibewithai

vibewithai.login(
    "sk-proj-xxxxxxx", # OpenAI
    "AIzaSyxxxxxxx",  # Google Gemini
    "xai-xxxxxxx",     # xAI (Grok)
    "sk-ant-xxxxxxx",  # Anthropic
)
```

Each credential is detected, verified with a lightweight health check, and the first working one becomes active. The rest stay ready as fallbacks.

Prefer keeping keys out of code:

```
export VIBEWITHAI_OPENAI_API_KEY=sk-proj-...
# conventional vendor variables work too
export ANTHROPIC_API_KEY=sk-ant-...
```

```
vibewithai.login() # env vars → stored credentials → local Ollama
```

Or store them once:

```
vibewithai login
```

2. Analyse

```
result = vibewithai.check()

print(result.score)      # 0-100
print(result.summary)
print(result.status)     # Status.WARNING
```

3. Read the result

Every command returns a `CommandResult` :

```
result = vibewithai.review()

for finding in result.findings:
    print(f'{{finding.location}} [{{finding.severity}}]  {{finding.title}}')
    print(f"    {{finding.detail}}")
    print(f"    from {{finding.source}} (confidence {{finding.confidence}})")
```

`source` tells you whether a finding came from deterministic static analysis

(`"ruff"`, `"bandit"`) or from a model (`"ai:openai"`) — so you always know how much to trust it.

Filter by severity

```
from vibewithai import Severity

blockers = result.findings_at_or_above(Severity.HIGH)  # most severe first
print(result.counts_by_severity)  # {'info': 2, 'low': 5, ..., 'critical': 0}
```

Use it as a gate

```
result = vibewithai.check()

if not result:                                # falsy only when the tool itself errored
    raise SystemExit(2)

if not result.passed:                          # False when blocking findings exist
    raise SystemExit(result.exit_code)
```

`ok` and `passed` mean different things on purpose. `ok` answers "did the tool

run?" — it is false only for an execution error. `passed` answers "is the code fine?" Use `passed` for gates; use `ok` to tell a broken tool from a bad verdict.

Export

```
result.to_dict()          # JSON-serialisable, safe to log
result.to_json(indent=2)
```

In CI

```
- run: pip install "vibewithai[analysis]"
- run: vibewithai check --fail-on high
  env:
    VIBEWITHAI_OPENAI_API_KEY: ${ secrets.OPENAI_API_KEY }
```

Exit codes: `0` clean or below threshold, `1` blocking findings, `2` execution error.

Without any AI

Local static analysis only — nothing leaves your machine:

```
config = vibewithai.load_config(offline=True)
vibewithai.set_active_config(config)
result = vibewithai.check()
```

Logging

Silent until you ask for it:

```
vibewithai.configure_logging(level="INFO")          # human readable
vibewithai.configure_logging(level="DEBUG", fmt="json") # for aggregators
```

Logs go to **stderr**, so piping machine-readable output on stdout stays clean.

Credentials are scrubbed from every record before emission.

Safety

- Nothing is written without explicit confirmation, and originals are backed up.
- Generated code is returned as data and never executed.
- Writes cannot escape the project root.

```
result = vibewithai.fix()
print(result.artifacts)          # proposed content, not yet on disk
result = vibewithai.fix(apply=True) # opt in to writing
```

API Reference

Everything below is importable directly from `vibewithai`. Names are resolved lazily, so importing the package costs nothing you do not use.

Results

CommandResult

Returned by every command.

ATTRIBUTE	TYPE	MEANING	
<code>command</code>	<code>str</code>	Name of the command	
<code>status</code>	<code>Status</code>	Terminal outcome	
<code>summary</code>	<code>str</code>	Human-readable summary	
<code>score</code>	<code>`int`</code>	<code>None`</code>	Quality score 0–100; <code>None</code> for non-assessment commands
<code>findings</code>	<code>list[Finding]</code>	Observations about the code	
<code>suggestions</code>	<code>list[Suggestion]</code>	Recommended actions	
<code>metrics</code>	<code>dict[str, float]</code>	Named measurements	
<code>confidence</code>	<code>float</code>	Overall confidence, 0.0–1.0	
<code>execution_time</code>	<code>float</code>	Wall-clock seconds	
<code>provider / model</code>	<code>`str`</code>	<code>None`</code>	What answered, if anything
<code>usage</code>	<code>TokenUsage</code>	Token accounting	
<code>artifacts</code>	<code>dict[str, str]</code>	Generated content, keyed by destination path	
<code>error</code>	<code>`dict`</code>	<code>None`</code>	Serialised error when <code>status</code> is <code>ERROR</code>

Properties

- `ok` — the command *ran* correctly (`SUCCESS` , `WARNING` or `SKIPPED`)

- `passed` — the command found nothing blocking (`SUCCESS` or `SKIPPED`); use this for quality gates
- `exit_code` — POSIX exit code for `status`
- `counts_by_severity` — `{severity: count}` histogram
- `highest_severity` — most severe finding, or `None`

Methods

- `findings_at_or_above(severity)` — filtered, most severe first
- `derive_status(blocking=Severity.HIGH)` — set `status` from findings; never overwrites `ERROR`
- `derive_score(baseline=100)` — set `score` from findings
- `add_finding` / `extend_findings` / `add_suggestion` / `add_metric`
- `to_dict(include_artifacts=True)` / `to_json(indent=2)`
- `CommandResult.from_error(command, error)` — classmethod

Results are iterable (over findings), sized, and truthy when `ok`.

Status

`SUCCESS` · `WARNING` · `FAILED` · `ERROR` · `SKIPPED`

`.exit_code` → 0 for success/warning/skipped, 1 for failed, 2 for error.

Severity

`INFO` · `LOW` · `MEDIUM` · `HIGH` · `CRITICAL`

Ordered by `.rank`, not alphabetically — `Severity.CRITICAL > Severity.HIGH` is `True`. Comparison against a non-`Severity` returns `NotImplemented`.

Finding

Frozen. `title`, `detail`, `severity`, `location`, `rule_id`, `source`, `confidence`, `fix`, `tags`. `.is_blocking` is `True` at `HIGH` or above.

`source` distinguishes deterministic analysis (`"ruff"`, `"bandit"`) from model output (`"ai:openai"`).

Suggestion

Frozen. `title`, `rationale`, `diff`, `location`, `effort` (`"low"` / `"medium"` / `"high"`).

SourceLocation

Frozen. `path`, `line`, `end_line`, `column`, `symbol`. Renders as `path:line`.

`SourceLocation.from_path(path, root=...)` relativises against the project root.

TokenUsage

`prompt_tokens`, `completion_tokens`, `cached_tokens`; `.total_tokens`.

Supports `+`.

Configuration

```
load_config(*, project_root=None, config_file=None, env=None,
            use_user_config=True, **overrides) -> Config
```

Resolves all five layers. See [configuration](#).

- `Config` — frozen; `.provider_settings(name)`, `.with_overrides(**changes)`, `.to_dict()`, `Config.default(project_root)`
- `get_active_config()` / `set_active_config(config)` — process-wide state
- `using_config(config)` — context manager, restores on exit

Credentials

Secret

Opaque wrapper. `str()`, `repr()`, f-strings and `%`-formatting all yield

`***REDACTED***`; pickling raises `TypeError`.

- `.reveal()` — the only path to the plaintext
- `.hint` — short non-reversible prefix for display, e.g. `'sk-p...'`
- Equality is constant-time and only against another `Secret`

CredentialStore

- `get(provider) -> Secret | None` — `env` → `vendor env` → stored file
- `require(provider) -> Secret` — raises `MissingCredentialsError`
- `set(provider, api_key)` / `delete(provider)` / `has(provider)`
- `providers() -> list[str]`
- `env_var_for(provider) -> str`

Writes `0600` files in the user config directory only.

Logging

```
configure_logging(*, level="WARNING", fmt="text", stream=None, propagate=False)
get_logger(name) -> logging.Logger
log_context(**fields)          # context manager
reset_logging()
```

Idempotent; defaults to stderr. Nothing is emitted until you call it.

`LogLevel` and `LogFormat` are `StrEnum`s.

Runtime

- `runtime_info()` -> dict — safe-to-publish diagnostics
- `ensure_supported_python(minimum=(3, 11))` — raises `UnsupportedPythonVersionError`

Exceptions

All descend from `VibeError`, which carries `.code`, `.message`, `.remediation`, `.context` and `.to_dict()`.

```
VibeError
├─ ConfigurationError
│   ├─ MissingConfigurationError      InvalidConfigurationError
│   └─ MissingCredentialsError        InvalidCredentialsError
├─ ValidationError                    ConfirmationRequiredError
├─ FileAccessError                    CacheError
├─ UnsupportedPythonVersionError
├─ ProviderError
│   ├─ ProviderNotFoundError          NetworkError
│   ├─ TimeoutError                  RateLimitError
│   └─ InvalidResponseError
├─ AnalysisError → AnalyzerNotAvailableError
└─ ScannerError                      ContextBuildError
```

Catch by category:

```
try:
    result = vibewithai.review()
except vibewithai.ProviderError as exc:      # any provider failure
    print(exc.code, exc.remediation)
except vibewithai.VibeError as exc:         # anything from this library
    print(exc.to_dict())
```

Utilities

Importable from `vibewithai.utils`:

- `read_text`, `atomic_write`, `safe_write`, `backup_file`, `is_within`,
`is_binary`, `iter_files`
- `estimate_tokens`, `fit_to_tokens`, `truncate`, `truncate_lines`, `mask_middle`
- `RetryPolicy`, `retry_call`, `is_retryable`

Configuration

Precedence

Highest wins:

- **Arguments** passed to the call — `load_config(offline=True)` , CLI flags
- **Environment variables** — `VIBEWITHAI_*`
- **Project config** — `vibewithai.toml` , `.vibewithai.toml` , or `[tool.vibewithai]` in `pyproject.toml`
- **User config** — `<user config dir>/config.toml`
- **Built-in defaults**

Nested tables merge key-by-key. Scalars and lists are replaced wholesale, so a project setting a list fully overrides the user's list rather than appending.

Inspect what actually resolved:

```
vibewithai config show
vibewithai config show --json
vibewithai config path      # where every file lives
```

Credentials are not part of this hierarchy

Credentials resolve only from:

- `VIBEWITHAI_<PROVIDER>_API_KEY`
- The provider's conventional variable (`OPENAI_API_KEY` , `ANTHROPIC_API_KEY` , `GEMINI_API_KEY` , ...)
- The user-level credential store (`vibewithai login`)

Project config files are never consulted for credentials, so a key cannot be committed to version control through one. See [SECURITY.md](#).

Project file

```
# vibewithai.toml
default_provider = "openai"      # null = use the first verified provider
offline          = false        # true = local static analysis only, no network
confirm_writes   = true         # false disables the overwrite guard (not advised)
max_context_tokens = 12000

# Top-level provider keys apply to every provider as defaults.
timeout          = 60.0
temperature      = 0.1

[providers.openai]
model            = "gpt-4o-mini"
timeout          = 30.0          # overrides the top-level default

[providers.ollama]
base_url         = "http://localhost:11434"
model            = "llama3.1"

[providers.ocil]
compartment_id   = "ocid1.compartment.oc1..." # unknown keys land in `extra`

[auth]
verify_on_login  = true          # live health check per credential
verify_timeout   = 10.0
fallback         = true          # switch providers on rate limit / outage
max_fallbacks    = 3
probe_unknown_keys = true        # identify keys matching no known prefix
allow_local      = true          # fall back to a local Ollama
preferred_order   = ["openai", "anthropic"]

[analysis]
analyzers        = ["ast", "ruff", "bandit", "radon"]
exclude          = [".git", "__pycache__", ".venv", "node_modules", "*_pb2.py"]
max_file_size    = 512000
max_files        = 5000
blocking_severity = "high"       # info | low | medium | high | critical
follow_symlinks   = false
workers          = 4             # omit to derive from CPU count

[cache]
enabled          = true
ttl              = 86400         # seconds; omit or null to never expire
```

```
max_entries = 5000
# directory = "~/cache/vibewithai"

[logging]
level = "WARNING"           # DEBUG | INFO | WARNING | ERROR | CRITICAL
format = "text"             # text | json
```

Equivalent inside `pyproject.toml` :

```
[tool.vibewithai]
default_provider = "openai"

[tool.vibewithai.analysis]
blocking_severity = "high"
```

A bare `pyproject.toml` with no `[tool.vibewithai]` table is ignored, so it never shadows a dedicated `vibewithai.toml`.

Environment variables

VARIABLE	MAPS TO
VIBEWITHAI_PROVIDER	default_provider
VIBEWITHAI_MODEL	model for the default provider
VIBEWITHAI_BASE_URL	endpoint override
VIBEWITHAI_OFFLINE	offline
VIBEWITHAI_CONFIRM_WRITES	confirm_writes
VIBEWITHAI_MAX_CONTEXT_TOKENS	max_context_tokens
VIBEWITHAI_PROJECT_ROOT	project_root
VIBEWITHAI_TIMEOUT	per-request timeout
VIBEWITHAI_MAX_RETRIES	retry attempts
VIBEWITHAI_TEMPERATURE	sampling temperature
VIBEWITHAI_MAX_OUTPUT_TOKENS	generation cap
VIBEWITHAI_LOG_LEVEL / VIBEWITHAI_LOG_FORMAT	logging
VIBEWITHAI_CACHE_ENABLED / _TTL / _MAX_ENTRIES	cache
VIBEWITHAI_VERIFY_ON_LOGIN / _VERIFY_TIMEOUT	auth
VIBEWITHAI_FALLBACK / _MAX_FALLBACKS	auth
VIBEWITHAI_PROBE_UNKNOWN_KEYS / _ALLOW_LOCAL	auth
VIBEWITHAI_PREFERRED_ORDER	comma-separated provider order
VIBEWITHAI_ANALYZERS / _EXCLUDE	comma-separated lists
VIBEWITHAI_MAX_FILE_SIZE / _MAX_FILES / _WORKERS	analysis
VIBEWITHAI_BLOCKING_SEVERITY	failure threshold

Path overrides (useful in CI and containers):

VARIABLE	EFFECT
VIBEWITHAI_CONFIG_DIR	User configuration directory
VIBEWITHAI_CACHE_DIR	Cache directory
VIBEWITHAI_CREDENTIALS_FILE	Exact credentials file path

Booleans accept `true/false`, `1/0`, `yes/no`, `on/off`. Lists accept comma-separated strings. An unrecognised `VIBEWITHAI_*` variable is ignored rather than rejected, because the same namespace also carries credentials.

From Python

```
from vibewithai import load_config, set_active_config, using_config

config = load_config(
    project_root="/path/to/project",
    offline=True,
    use_user_config=False,      # ignore personal defaults – do this in CI
)

set_active_config(config)      # install process-wide

with using_config(config):     # or scope it to a block
    ...
```

Config is frozen. Derive variants with `with_overrides()`:

```
strict = config.with_overrides(confirm_writes=True, offline=False)
```

Reproducible CI

```
env:
  VIBEWITHAI_OPENAI_API_KEY: ${ secrets.OPENAI_API_KEY }
  VIBEWITHAI_LOG_FORMAT: json
  VIBEWITHAI_CACHE_DIR: .cache/vibewithai
```

Pass `use_user_config=False` (library) so a developer's personal defaults can never influence a CI run.

Architecture

Layering

The dependency graph is strictly bottom-up. Each layer may import from the layers below it and never from those above.

cli	thin renderer		
commands	check, review, fix ...		
context	scanner	analysis	providers
config	utils		
core	logging, cache, runtime		
exceptions	models		

`exceptions` and `models` have no internal dependencies at all, which is what lets every other layer raise and return without risking a cycle.

Why it matters: an upward import is always a design error, and the cheapest way to catch one is to make the rule explicit. If a lower layer needs behaviour from a higher one, invert the dependency with a `Protocol` rather than importing.

Key decisions

Lazy imports at the package root

`import vibewithai` binds no submodules. The root module holds a name→module table and a [PEP 562](#) `__getattr__` that

imports on first attribute access and caches the result in the module globals.

Measured cold import: **~2 ms** (budget 100 ms, enforced by

`tests/unit/test_package.py`). Touching `vibewithai.Severity` loads

`vibewithai.models` and nothing else; `httpx`, `rich`, `typer` and the config subsystem stay unloaded until something actually needs them.

The consequence developers feel: the SDK is usable inside pre-commit hooks and editor plugins, where a 300 ms import would be noticed on every keystroke.

Structured results, not printed text

Every command returns a `CommandResult`. The CLI is one renderer among many; the same call works from CI, a notebook, or a web service. `CommandResult` carries `status`, `score`, `summary`, `findings`, `suggestions`, `metrics`, `confidence`, `execution_time`, `usage` and `provider`, and serialises with `to_dict()`. `Status` maps to POSIX exit codes so CI gating needs no translation layer, and `Severity` overrides all four comparison operators because `StrEnum` would otherwise order `critical` before `high` alphabetically and silently invert every quality gate.

Protocols over base classes

`Cache` is a `Protocol`, not an ABC. `DiskCache` and `NullCache` satisfy it without inheriting anything, and a user's Redis-backed cache would too. The same approach applies to providers and analyzers.

`NullCache` is the Null Object pattern: call sites depend on the interface unconditionally instead of guarding every access with `if cache is not None`.

Immutable, self-validating configuration

Every settings object is a frozen dataclass that validates in `__post_init__`.

An invalid configuration cannot be constructed, so no consumer downstream has to re-check values. Changes go through `with_overrides()`, which returns a new instance.

Configuration resolves through five layers (explicit arguments → environment → project file → user file → defaults). Credentials are deliberately *outside* this hierarchy — see below.

Credentials are structurally separate from configuration

`Config` has no credential fields, which makes `config.to_dict()` inherently safe

to print, log or serialise. Credentials resolve lazily through `CredentialStore`, which reads only from the environment and the user-level store — never from project files. A key therefore cannot reach version control through a committed config file.

`Secret` makes the plaintext reachable only via an explicit `.reveal()`.

`str()`, `repr()`, `__format__` and `%`-formatting all yield a placeholder, and

`__reduce__` refuses to pickle. Independently, a logging filter scrubs

credential-shaped text from every record before emission, so the two mechanisms would both have to fail for a key to leak.

Failure isolation

Different subsystems have different failure semantics, chosen deliberately:

SUBSYSTEM	ON FAILURE	WHY
Cache	Degrade to a miss, log at debug	A broken cache must never break a build
Provider	Raise a typed error with remediation	The user must know and can act
Filesystem write	Refuse, raise	Data loss is unacceptable
Analyzer backend missing	Skip with a finding	Partial analysis beats no analysis

Safety invariants enforced in code

- Writes outside the project root are refused (`is_within` check on every path).
- Existing files are never overwritten without explicit caller confirmation, and are snapshotted first.
- Writes are atomic: temp file → `fsync` → rename. An interrupted run cannot truncate a source file.
- AI-generated code is never executed, `eval`-ed or imported.

Retry policy

Only genuinely transient failures are retried — timeouts, connection errors, and

`429 / 5xx`. Retrying a `401` cannot succeed and only delays a clear error.

Backoff uses **full jitter** (uniform over `[0, delay]`). Without jitter, N

workers that hit the same rate limit retry in lockstep and re-trigger it. A

server-supplied `Retry-After` always wins over the computed delay.

Module map

MODULE	RESPONSIBILITY
<code>exceptions</code>	Typed error hierarchy with stable codes and remediation hints
<code>models</code>	<code>CommandResult</code> , <code>Finding</code> , <code>Suggestion</code> , <code>Severity</code> , <code>Status</code>
<code>core.logging</code>	Structured logging, context binding, secret redaction
<code>core.cache</code>	<code>Cache</code> protocol, <code>DiskCache</code> , <code>NullCache</code>
<code>core.runtime</code>	Version guards, optional-backend detection, diagnostics
<code>config.models</code>	Frozen, validated settings dataclasses
<code>config.loader</code>	Five-layer configuration resolution
<code>config.paths</code>	Platform directories, project-root discovery
<code>config.secrets</code>	<code>Secret</code> , <code>CredentialStore</code>
<code>config.session</code>	The single piece of ambient state, lock-guarded
<code>utils.fs</code>	Atomic, confirmed, contained writes; guarded reads
<code>utils.text</code>	Token estimation and code-aware truncation
<code>utils.retry</code>	Retry policy with full-jitter backoff
<code>cli</code>	Argument parsing and rendering only — no logic

Concurrency

The library holds exactly one piece of mutable process state: the active configuration in `config.session` , guarded by an `RLock` . It is centralised there so that no other module needs a global.

Log context binding uses `contextvars` , making it coroutine- and task-local.

Note that plain `threading.Thread` does **not** inherit context — use `contextvars.copy_context()` to propagate bindings to a worker thread.

Developer Guide

For contributors and for anyone extending the SDK. Setup and house rules live in [CONTRIBUTING.md](#); this covers extension points and the reasoning behind them.

The layering rule

Imports flow strictly bottom-up:

```
exceptions, models → core → config, utils → providers, scanner, analysis
                        → context → commands → cli
```

An upward import is a design error. Invert it with a `Protocol` instead. A function-scope import added to "break a cycle" is a warning sign — the one legitimate use of deferred imports here is lazy loading for import speed, which is confined to the package root and a few documented call sites.

Adding a provider

A provider is one file plus one registry entry. If you find yourself editing anything else, the abstraction needs fixing first — please open an issue.

Every provider implements the same interface:

```
class Provider(Protocol):
    name: str

    def authenticate(self, secret: Secret) -> None: ...
    def health_check(self) -> bool: ...
    def chat(self, messages: Sequence[Message], **options) -> Completion: ...
    def generate(self, prompt: str, **options) -> Completion: ...
    def embeddings(self, texts: Sequence[str]) -> list[list[float]]: ...
```

Rules:

- **Wrap every third-party exception.** No `httpx.HTTPError` may escape your module. Translate to `NetworkError`, `TimeoutError`, `RateLimitError`,

`InvalidCredentialsError` or `InvalidResponseError` so the retry layer and the CLI can act on it.

- **Map status codes correctly.** `401 / 403` → `InvalidCredentialsError` (never retried). `429` → `RateLimitError`, carrying `retry_after` when the response provides it. `5xx` → `ProviderError` with `status_code` in `context`.
- **Never log the key.** Take a `Secret` and call `.reveal()` at the point the header is built, not earlier.
- **Reuse the HTTP client.** One pooled `httpx.Client` per provider instance; creating one per request destroys connection reuse.
- **Register the key pattern centrally.** Detection lives in the registry, never scattered across modules.

Then: unit tests with stubbed HTTP (`respx`) — never a live call — and an entry in `docs/configuration.md` and the README provider table.

Adding an analyzer

Analyzers turn tool output into `Finding` objects:

```
class Analyzer(Protocol):
    name: str
    def available(self) -> bool: ...
    def analyze(self, target: Path, config: Config) -> list[Finding]: ...
```

- Return `available()` is `False` rather than raising when the backend is not installed; the pipeline skips you and continues. A missing optional tool must never fail the whole run.
- Set `source` to your analyzer name and `confidence=1.0` for deterministic results.
- Map the tool's severity onto ours honestly. Inflating severity makes quality gates useless.
- Use `SourceLocation.from_path(path, root=config.project_root)` so paths are displayed relative to the project.

Error design

Every error must be catchable by category, carry a stable `code`, and offer an

actionable remediation :

```
class ProviderQuotaExceededError(ProviderError):
    """The account's quota is exhausted."""

    code = "quota_exceeded"

    def __init__(self, provider: str, *, reset_at: str | None = None) -> None:
        super().__init__(
            f"Provider {provider!r} quota is exhausted.",
            provider=provider,
            remediation="Add billing credit, or switch provider with --provider.",
            context={"reset_at": reset_at},
        )
```

Codes are part of the public contract — never change what one means. `context`

must never contain a credential; run values through

`vibewithai.config.redact` first if there is any doubt.

Working with results

Build results incrementally, then derive the verdict:

```
result = CommandResult(command="check")
with result_timer() as timer:
    result.extend_findings(run_analysis())
timer.apply(result)

result.derive_status() # from findings; never overwrites ERROR
result.derive_score()
```

Never return a bare string, and never `print()` from anything below the CLI

layer — the ruff config blocks it.

Import-time cost

`import vibewithai` must stay under 100 ms;

`tests/unit/test_package.py` enforces it and also asserts that `httpx`, `rich`,

`typer` and `vibewithai.config` are *not* imported as a side effect.

If you add a public symbol, add one line to `_EXPORTS` and one to `__all__` in

`src/vibewithai/__init__.py`. A test keeps the two in sync. Do not add a

module-level import there.

Testing conventions

- Never call a real provider. Use stubbed HTTP or the mock provider.
- The autouse `isolate_environment` fixture redirects config, cache and credential paths into `tmp_path` and clears inherited credential variables — do not bypass it.
- Inject clocks and sleeps (`retry_call(..., sleep=lambda _: None)`) so tests stay fast and deterministic.
- Test names should state the behaviour, not the method:
`test_expired_entries_are_treated_as_misses_and_evicted` .
- When a test encodes a subtle invariant, add a one-line comment saying what regression it guards against.

Release

- Update `__version__` in `src/vibewithai/__about__.py` (single source of truth).
- Move `[Unreleased]` to a dated version heading in `CHANGELOG.md` .
- `python -m build && twine check --strict dist/*`
- Verify the wheel in a clean venv: import, then `vibewithai --version` .
- Tag `vX.Y.Z` and push; CI publishes.

Best Practices

Run `vibewithai check --offline` in CI first: it is deterministic, needs no credential, and turns static findings into a quality gate. Use `--format json` when another tool consumes the result and set `--fail-on high` (or a stricter level) deliberately.

Keep credentials in environment variables or the credential store created by `vibewithai login`; never place them in project configuration. Review generated artifacts before applying them, then use `--yes` only when you intend to write.

Use `--changed-only` in pull-request jobs to analyse the diff quickly, and run a full `check` periodically to catch cross-project structure issues.

Troubleshooting

Start here:

```
vibewithai info           # versions, platform, available backends
vibewithai config show    # what configuration actually resolved
vibewithai config path    # where each file lives
vibewithai config providers # which credentials are visible
vibewithai --verbose <command> # debug logging on stderr
```

Every error raised by this library carries a remediation hint. The CLI prints it

as `hint: ;` in Python it is `error.remediation`.

MissingCredentialsError

No credential resolved for the provider.

```
vibewithai config providers    # what is currently visible
export VIBEWITHAI_OPENAI_API_KEY=sk-proj-...
# or
vibewithai login
```

Most common cause: the variable is set in a different shell, or set *after* the process started. Environment is read at call time, not import time — but a process that has already started will not see a change made in your terminal.

InvalidCredentialsError

The provider rejected the key (401/403). Possible reasons:

- The key was revoked or has expired
- The key lacks the required scope for the model you requested
- You pasted a *project* id or organisation id instead of the key
- Trailing whitespace or quotes were included

`vibewithai config providers` shows a short hint of the stored key so you can confirm which one is in use without printing it.

RateLimitError

Automatic fallback normally handles this by switching to your next verified provider. If it surfaces:

- You have only one provider configured — add another to `login()`
- Fallback is disabled — set `auth.fallback = true`
- All providers are limited — reduce `analysis.workers` or enable caching

TimeoutError / NetworkError

```
[providers.openai]  
timeout = 120.0
```

Behind a corporate proxy, `httpx` honours `HTTPS_PROXY` and `NO_PROXY`. For air-gapped environments use `offline = true`.

ConfirmationRequiredError

Working as intended: a command tried to modify a file and you had not authorised it.

```
vibewithai.fix(apply=True)    # library
```

```
vibewithai fix --yes         # CLI
```

Review `result.artifacts` first. The original file is backed up as `<name>.vibe.bak` before replacement.

FileAccessError: destination escapes the project root

A write target resolved outside your project. This guard is not configurable — it is what prevents a path-traversal write from AI or user input. Run the command from the correct project root, or set `VIBEWITHAI_PROJECT_ROOT`.

AnalyzerNotAvailableError

```
pip install "vibewithai[analysis]"
```

Or drop the backend from `analysis.analyzers`. Analysis continues with whatever

is available rather than failing outright.

Configuration seems to be ignored

```
vibewithai config show
```

The `sources` list at the bottom shows which files contributed, least significant first. Common causes:

- The file is not at the detected project root — check `vibewithai config path`
- You used `[tool.vibewithai]` in a file other than `pyproject.toml`
- An environment variable is overriding it (environment beats project config)
- A typo in the key name — unknown keys are ignored, not rejected

```
Configuration file ... contains invalid TOML
```

The message includes the line number. A frequent mistake is an unquoted string:

```
default_provider = openai      # wrong
default_provider = "openai"    # right
```

Slow runs

```
vibewithai cache stats
```

- Ensure `cache.enabled = true`
- Raise `analysis.workers`
- Narrow `analysis.exclude` so vendored trees are pruned
- Lower `max_context_tokens` to reduce prompt size

Stale or corrupt cache

Corrupt entries are detected and evicted automatically. To reset manually:

```
vibewithai cache clear
vibewithai cache clear --expired-only
vibewithai cache clear --namespace check
```


Warning: credentials file is accessible to other users

```
chmod 600 "${vibewithai config path | awk '/credentials/{print $2}'}"
```

Permission denied writing credentials

The user config directory is not writable — common in containers. Use environment variables instead, or point elsewhere:

```
export VIBEWITHAI_CONFIG_DIR=/writable/path
```

Still stuck?

Open an issue with:

```
python -c "import vibewithai, json; print(json.dumps(vibewithai.runtime_info(), indent=2))"
```

plus a minimal reproduction and the `--verbose` output. `runtime_info()` excludes environment variables and home paths, so it is safe to post publicly.

For security issues, follow [SECURITY.md](#) instead.

Frequently Asked Questions

Is this just a wrapper around an LLM API?

No. Local static analysis runs first — AST, Ruff, Bandit, Radon — and the model is consulted only for questions those tools cannot answer. That makes runs cheaper, faster, and deterministic where determinism is possible. Findings carry a `source` field so you can always tell which produced what.

Do I have to pick a provider?

No. Pass whatever keys you have to `login()` in any order; the provider is detected from the key shape and confirmed with a health check. If the active one is rate limited or down, the SDK switches to the next verified provider automatically.

What if I have no API key at all?

`login()` with no arguments checks your environment, then your stored credentials, then a locally running Ollama. With none of those, use `offline=True` for local static analysis only.

Does my source code get uploaded?

Only a bounded, relevant slice — the file in question plus related imports and symbols — and only for commands that need a model. Nothing is sent for local analysis. Set `offline = true` to guarantee nothing leaves your machine. This is a data-egress decision only you can make; see [SECURITY.md](#).

Will it modify my files?

Not without explicit confirmation. Commands that would write raise `ConfirmationRequiredError` unless you pass `apply=True` (library) or `--yes` (CLI), and the original is backed up first. Writes outside the project root are refused outright.

Does it ever run AI-generated code?

Never. Generated code is returned to you as data. It is not executed, `eval`-ed, or imported.

Can my API key leak into logs?

Two independent mechanisms would both have to fail. `Secret` masks itself in every stringification path, and a logging filter scrubs credential-shaped text from every record before emission. Credentials are also absent from the `Config` object entirely, so `config show` cannot print one.

Can a key end up in git?

Not through this library. Credentials are read only from environment variables and the user-level store; project config files are never consulted for them.

Why is `import vibewithai` so fast?

It binds no submodules. A PEP 562 `__getattr__` imports each symbol's module on first access. Cold import measures ~2 ms, and a test fails the build if it exceeds 100 ms.

What is the difference between `ok` and `passed` ?

`ok` means the command *ran* correctly. `passed` means it found nothing blocking.

A review that runs perfectly and finds three critical bugs is `ok` but not `passed`. Use `passed` for CI gates.

How is `score` calculated?

100 minus a severity-weighted penalty: critical 25, high 12, medium 5, low 2, info 0 — clamped to `[0, 100]`. Deliberately simple and stable, because a score that shifts between releases for unchanged code is useless for tracking a trend.

Why does `Severity.CRITICAL > Severity.HIGH` work?

Because all four comparison operators are overridden to use severity rank.

`StrEnum` would otherwise compare alphabetically and put `critical` below `high`, silently inverting every quality gate.

Can I use it without the CLI?

Yes — the CLI is a thin renderer over the same public functions. Everything works

from a library call, and results are plain data.

Can I add my own provider or analyzer?

Yes. Providers implement a protocol and register with the registry; adding one should be a single new file plus a registry entry. See the [developer guide](#).

Which Python versions are supported?

3.11+. CI tests 3.11, 3.12 and 3.13 on Linux, macOS and Windows.

Is it safe to run in CI?

Yes, and it is designed for it. Pass `use_user_config=False` so a developer's personal settings cannot affect the run, supply keys via environment variables, and gate on the exit code.

How do I report a security issue?

Privately, per [SECURITY.md](#) — please do not open a public issue.

Security Policy

Supported versions

VERSION	SUPPORTED
0.1.x	

Until 1.0, security fixes land on the latest minor release only.

Reporting a vulnerability

Please do not open a public issue for security problems.

Report privately via GitHub's [private vulnerability reporting](https://github.com/ragulraj-d/vibewithai/security/advisories/new).

Please include:

- A description of the issue and its impact
- Steps to reproduce, or a proof of concept
- Affected version(s) and platform
- Any suggested mitigation

What to expect: acknowledgement within 3 working days, an initial assessment within 10 working days, and coordinated disclosure once a fix is available. We will credit you in the advisory unless you prefer otherwise.

Security model

What this library protects

Credential confidentiality. API keys are wrapped in `Secret`, which masks itself in `str()`, `repr()`, `__format__` (so f-strings are safe) and refuses to be pickled. The plaintext is reachable only via an explicit `.reveal()` call.

A logging filter independently scrubs credential-shaped text from every record before emission, so even a careless `logger.debug("headers=%s", headers)` cannot leak a key.

No credentials in version control. Credentials are read only from environment

variables and the user-level credential store. The project configuration loader does not consult credential keys at all, so a key cannot be committed by accident. The credential file is written with `0600` inside a `0700` directory, and the library warns if those permissions loosen.

No implicit code execution. AI-generated code is *never* executed, `eval`-ed, `exec`-ed or imported. It is returned to the caller as data.

No implicit writes. Any command that would modify the workspace raises `ConfirmationRequiredError` unless the caller explicitly opts in, and existing files are snapshotted before replacement. Every destination path is verified to resolve inside the project root, which blocks path traversal via AI- or user-supplied paths.

Atomic writes. Files are written to a temporary file, `fsync`-ed, then atomically renamed, so an interrupted run cannot truncate a source file.

Input validation. Provider names, API key shapes and all configuration values are validated before use. Malformed input fails fast with a clear message rather than reaching a network client or the filesystem.

What it does not protect against

- **A local attacker running as your user.** The credential file is protected by filesystem permissions, not encryption. Anyone able to run code as you can read it, and can equally read your shell history and environment. For stronger isolation, keep keys in an OS keychain or secret manager and expose them through `VIBEWITHAI_<PROVIDER>_API_KEY`, which takes precedence over the file.
- **Your source code reaching a third party.** Any AI command sends a bounded, relevant slice of your code to the configured provider. This is the library's purpose, but it is a data-egress decision only you can make. Use `offline=True` to restrict operation to local static analysis, and review your provider's data retention terms before enabling AI features on proprietary code.
- **Provider-side compromise.** We cannot vouch for the security of third-party AI services.
- **Correctness of AI output.** Suggestions are probabilistic. Review every

proposed change before applying it, particularly security-relevant fixes.

Findings carry a `confidence` score and a `source` field so AI output is always distinguishable from deterministic static analysis.

Hardening recommendations

- Prefer environment variables in CI over the credential file.
- Set `offline = true` for repositories whose source must not leave your network.
- Use a per-project API key with the narrowest available scope so a leak is cheap to revoke.
- Review `vibewithai fix` and `vibewithai refactor` diffs before applying, and keep a clean working tree so changes are easy to inspect and revert.
- Pin the version in production tooling and review the changelog before upgrading.

Our own supply chain

- Runtime dependencies are kept minimal and pinned to compatible ranges.
- CI runs Bandit for static security analysis and fails the build on findings.
- Releases are built with `hatchling` from a clean checkout and published from a tagged commit.

Migration Guide

`vibewithai` has no legacy configuration format. Start with `vibewithai init`, commit only the generated project configuration, and keep credentials outside the repository. Existing linters can remain in place: `vibewithai` invokes Ruff, Bandit, and Radon as optional local backends and normalises their findings. For CI, begin with `vibewithai check --offline --format json`; once that output is stable, choose a `--fail-on` severity and add provider-backed review jobs only where AI reasoning is useful.

Changelog

All notable changes to this project are documented here.

The format follows [Keep a Changelog](#), and

this project adheres to [Semantic Versioning](#).

[Unreleased]

[0.1.1] — 2026-07-30

Security

- ****API keys for Groq, Fireworks, Perplexity and GitHub fine-grained tokens were not redacted from log output.**** The vendor pattern list had drifted from the provider catalogue, so `gsk_...`, `fw_...`, `pplx-...` and `github_pat_...` credentials could appear in plaintext in logs and CI transcripts, and were missed by the hard-coded-secret analyzer. All four are now masked, and a behavioural test synthesises a key from every provider's own declared prefix and asserts it is redacted — so a provider can no longer be added without redaction coverage.

Fixed

- ****Conventional vendor environment variables were ignored for 12 of 16 providers.**** The credential store kept its own four-entry table while the provider catalogue listed variables for every provider, so `XAI_API_KEY`, `GROQ_API_KEY`, `MISTRAL_API_KEY`, `HF_TOKEN`, `GITHUB_TOKEN`, `DEEPSEEK_API_KEY`, `COHERE_API_KEY`, `TOGETHER_API_KEY`, `PERPLEXITY_API_KEY`, `OPENROUTER_API_KEY`, `FIREWORKS_API_KEY` and `AZURE_OPENAI_API_KEY` were documented as honoured but silently did nothing. The catalogue is now the single source of truth, with a test asserting the two can never diverge again.

Changed

- **Removed AWS Bedrock, Google Vertex AI and OCI Generative AI.** They were

advertised but had no working authentication, requiring cloud-native request signing rather than bearer tokens. The catalogue is now 16 providers that all work with an API key alone. Reach those models via OpenRouter meanwhile.

- Rewrote the project description for the package index: structured overview, full command reference, and enterprise sections on security, data governance, CI/CD and extensibility.

Removed

- Dead code: `ProviderSpec.openai_compatible` (written by every spec, never read), `ProviderSpec.supports_embeddings`, `AnalysisReport.by_severity`, and the now-unused `AuthStyle.SIGV4`.

[0.1.0] — 2026-07-30

Initial public release.

Fixed

- `--format json` **output was corrupted**. Machine-readable output was printed through `rich`, which soft-wraps at the terminal width and inserted newlines inside JSON strings — `vibewithai check --format json | jq` failed. Non-terminal formats now write straight to stdout.
- **The error handler could crash**. `commands.engine` logged `extra={"message": ...}`; `message` is a reserved `LogRecord` attribute, so `logging` raised `KeyError` and turned a handled error into an unhandled one.
- **A target outside the project was silently ignored**, widening the scope to the whole project. A mistyped path now raises `ContextBuildError` with the reason.
- `CommandResult.ok` **wrongly excluded** `FAILED`. A run that completed and found blocking issues is a successful analysis with a bad verdict, not a failed run.
`ok` is now true for everything except `ERROR`; `passed` remains the quality gate.
- `optimize` **and** `deploy` **rejected** `--offline` **and** `--fail-on`, unlike every other analysis command.
- `vibewithai init` **aborted** when Enter was pressed at the API-key prompt instead of skipping.
- Static-analysis and type-checking cleanups across the analysis, authentication, reporting, context, coverage, and OpenAI-compatible provider paths. Malformed

completion choices now produce a domain error instead of reaching response parsing unchecked.

- Secret detection now coalesces overlapping pattern matches, so one credential is reported once at its strongest severity instead of producing duplicates.

Added

- **Plugin registry** (`vibewithai.plugins`) — one facade for providers, analyzers, and report renderers, with explicit entry-point discovery through the ``vibewithai.plugins`` group.
- **Release and adoption material** — OIDC-based PyPI release workflow, GitLab, Jenkins and Azure DevOps CI examples, plus best-practice, migration, and release-notes guides.
- **Changed-only scanning** — `vibewithai check --changed-only` limits project analysis to Python paths changed from `HEAD` .
- **Examples and benchmarks** — minimal FastAPI, Flask, and data-science examples, plus a repeatable import and cache-speed benchmark script.

Added — Phase 1: foundation layer

- **Exception hierarchy** (`vibewithai.exceptions`) — a single `VibeError` root with stable machine-readable `code` values and an actionable `remediation` hint on every error. Covers configuration, credentials, validation, filesystem, provider, rate-limit, timeout and analysis failures.
- **Structured result models** (`vibewithai.models`) — `CommandResult` , `Finding` , `Suggestion` , `SourceLocation` , `TokenUsage` , plus `Status` and `Severity` enums with rank-based ordering and POSIX exit-code mapping. Results carry `score` and `metrics` and serialise via `to_dict()` / `to_json()` .
- **Structured logging** (`vibewithai.core.logging`) — text and JSON formatters, `contextvars` -based field binding via `log_context()` , and a `RedactingFilter` that scrubs credential-shaped values from messages *and* structured fields.

A `NullHandler` is installed at import, so the library is silent until an application calls `configure_logging()` .
- **Layered configuration** (`vibewithai.config`) — immutable, self-validating settings dataclasses resolved from explicit arguments, `VIBEWITHAI_*`

environment variables, project config (`vibewithai.toml` , `.vibewithai.toml` , `[tool.vibewithai]`), user config, and built-in defaults.

- **Credential handling** (`vibewithai.config.secrets`) — an opaque `Secret` type that masks itself in `str()` , `repr()` , f-strings, and refuses to pickle; plus a `CredentialStore` writing `0600` files in the user config directory only, honouring conventional per-provider environment variables.
- **Cache layer** (`vibewithai.core.cache`) — a `Cache` protocol with `DiskCache` (content-addressed, atomic writes, TTL expiry, oldest-first eviction) and `NullCache` . Cache failures always degrade to a miss, never an error.
- **Safe filesystem utilities** (`vibewithai.utils.fs`) — atomic writes, pre-write backups, project-root containment checks, size- and encoding-guarded reads, and a pruning directory walker.
- **Retry policy** (`vibewithai.utils.retry`) — exponential backoff with full jitter that retries only genuinely transient failures and honours server-supplied `Retry-After` .
- **Token budgeting** (`vibewithai.utils.text`) — dependency-free conservative token estimation and code-aware truncation.
- Packaging, CI workflow, pre-commit configuration and documentation set.

Security

- Credentials are resolved only from the environment and the user-level store — never from project configuration files — so a key cannot reach version control through a committed config file.
- All log records pass through redaction before emission.
- Writes are refused outside the project root and, for existing files, without explicit caller confirmation.

Roadmap

- **Phase 2** — universal `login()` : provider registry, key-shape detection, live verification, and automatic runtime fallback across providers.
- **Phase 3** — provider implementations for the full supported set.
- **Phase 4** — project scanner and static-analysis pipeline (AST, Ruff, Bandit, Radon, LibCST) with incremental caching.

- **Phase 5** — context builder and the command layer (`check` , `review` , `fix` , `security` , `document` , `test` , ...).
- **Phase 6** — CLI, report renderers (JSON/Markdown/HTML), plugin registry, and CI/CD integration templates.

[Unreleased]: <https://github.com/ragulraj-d/vibewithai/commits/main>