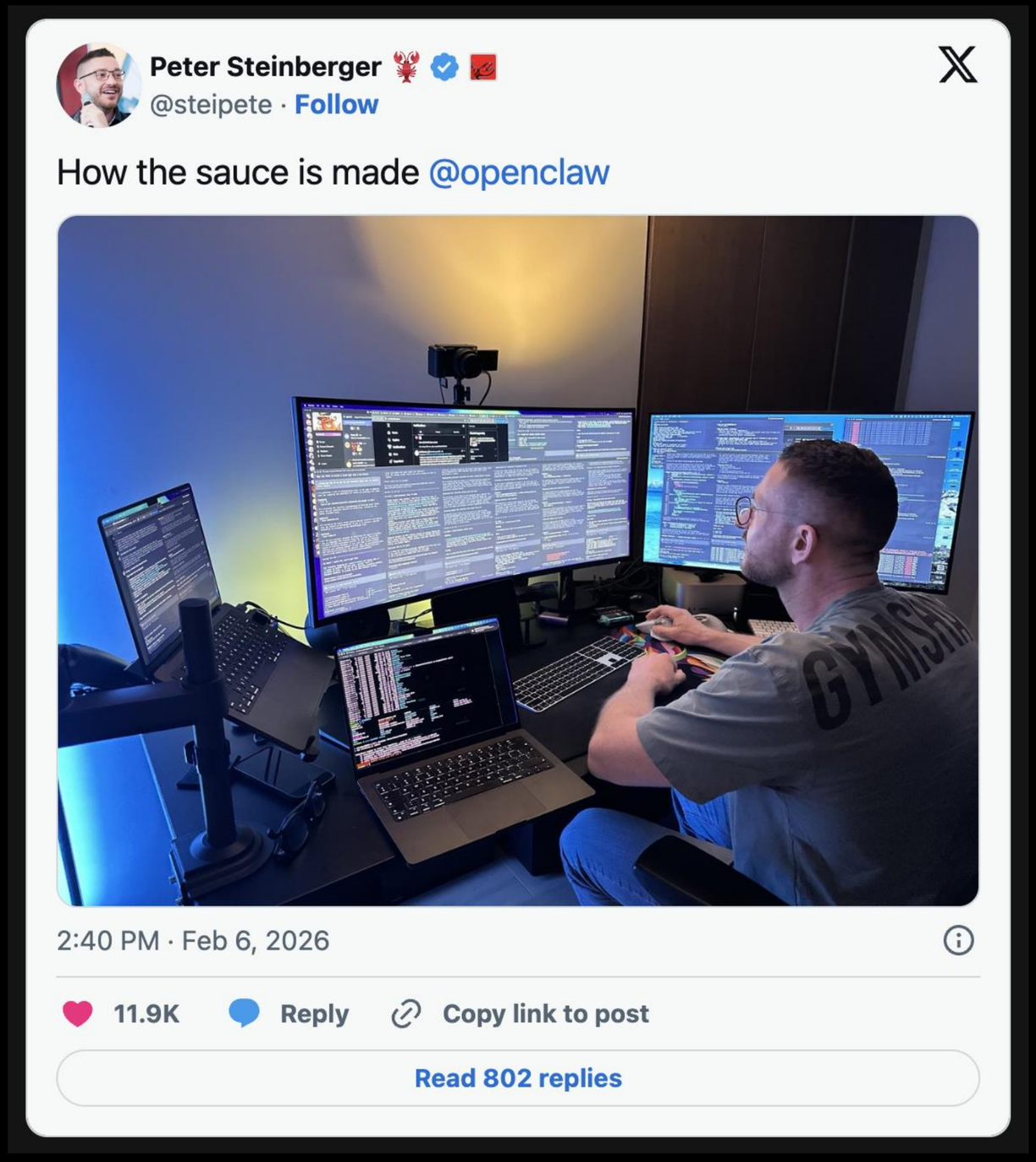
INSOMNIA

Secure Network OS

AIで作るNetwork OS

Kunihiro Ishiguro 2026/10/26



Xに投稿されたOpenClaw開発者による並列AI開発の姿

同時に4〜5のAIセッションを同時に立ち上げる開発スタイル

背景にAIコーディングエージェントの性能が飛躍的に上がったこと

git worktreeを活用した同一レポジトリ内での並行開発の
ノウハウが広まったこと

いつかこういう日が来るんじゃないかと思っていたが

ついに来た

ontology-driven · agent-ready

AI, Ontology and Networking.

zebra-rs is a BGP, OSPF, and IS-IS routing stack with SRv6, SR-MPLS, L3VPN, and EVPN extensions, written from scratch in Rust. It exposes the network to AI agents as an ontology — nodes, links, and the vocabulary that connects an attribute to an intent — over a native MCP server, so an agent can reason about your fabric instead of guessing at it. Memory-safe, async to the core, idempotent by design.

Get started → Read the docs

```
$ curl -fsSL https://zebra.rs/install.sh | bash
```
COPY

⚡ 14ms commit 🦀 100% safe Rust ◰ AGPLv3 🔌 MCP native

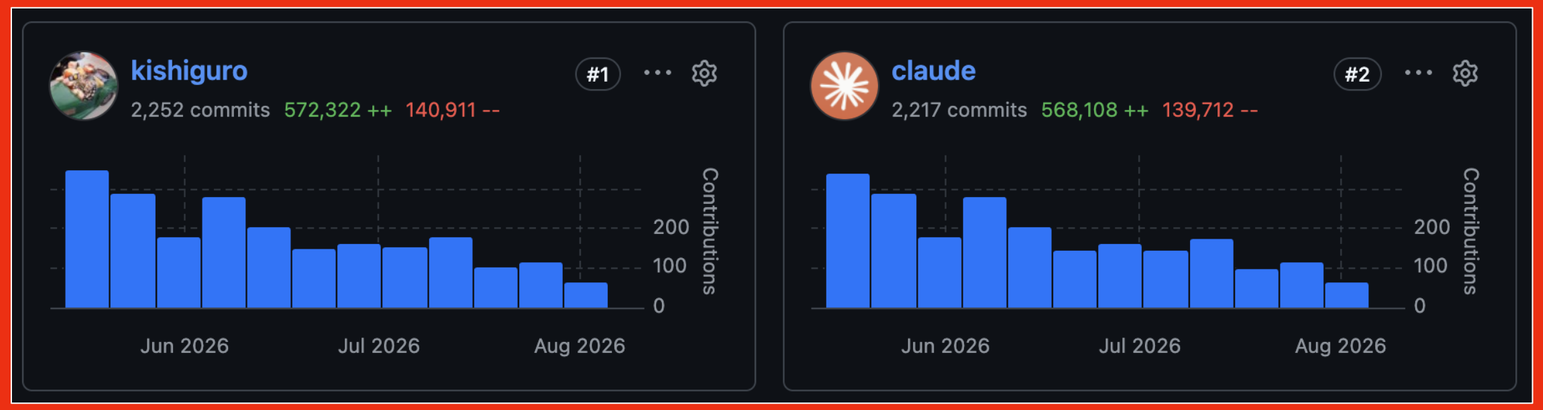
zebra@edge-01 — zsh — 82×24 ● live

```
zebra-rs version v26.9.2 (9e48ac5)
Build Date: 2026-06-22 23:51:42 UTC
zebra ❯ configure
zebra # set router bgp global as 65501
zebra # set router bgp neighbor 10.0.0.2 remote-as 65501
zebra # set router bgp neighbor 10.0.0.2 afi-safi vpnv4 enabled true
zebra # show
+router {
+  bgp {
+    global {
+      as 65501;
+    }
+    neighbor 10.0.0.2 {
+      afi-safi vpnv4 {
+        enabled true;
+      }
+      remote-as 65501;
+    }
+  }
+}
zebra # commit
```

TERMINAL

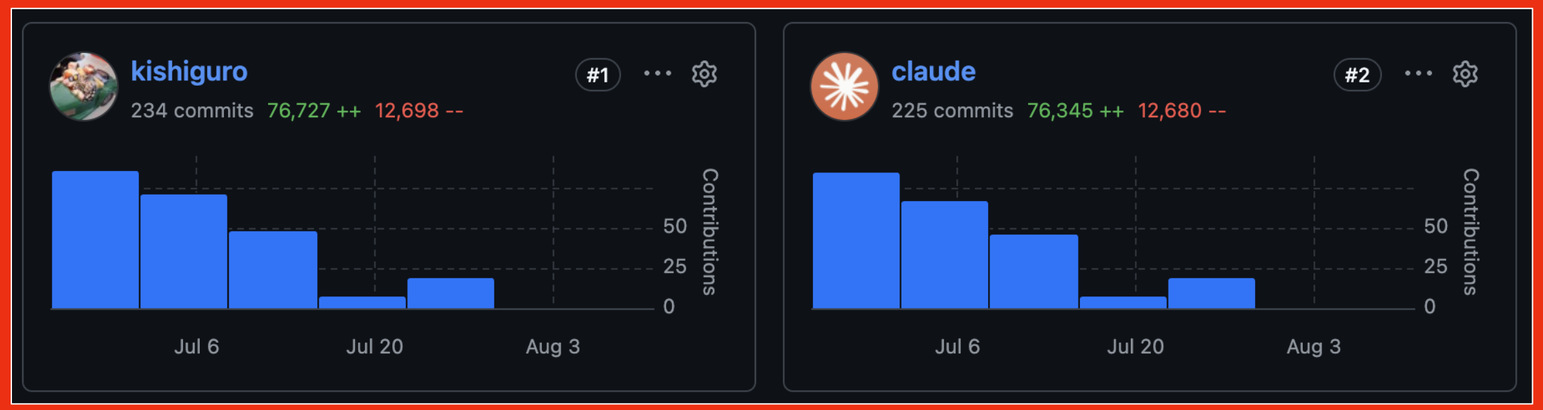
2026年5月14日〜8月7日 - リリース前後の約3ヶ月間の追加コード行数

zebra-rs



572,322行

cradle-rs



76,727行

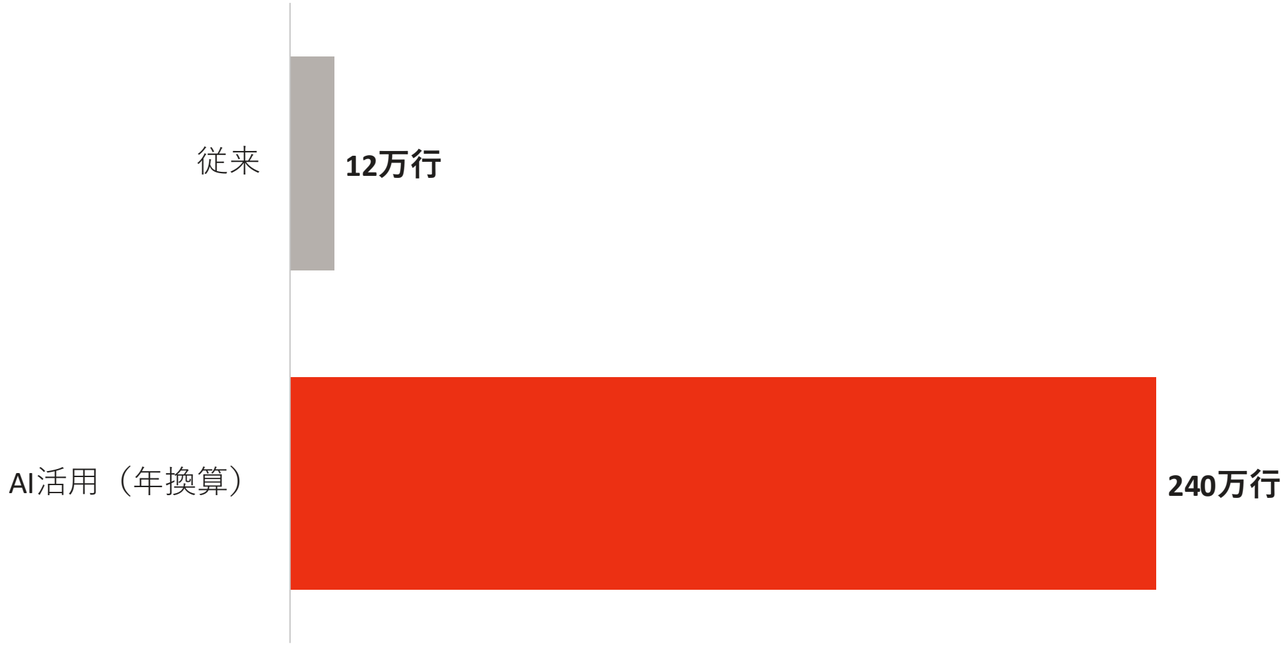
合計(約3ヶ月)

649,049行

生産性

AI活用で、年間のコード生産量は当人比従来の20倍に

年間のコード行数



20倍

年間のコード行数（AI活用 ÷ 従来）

AI活用：3ヶ月で約60万行（年換算 **約240万行**）

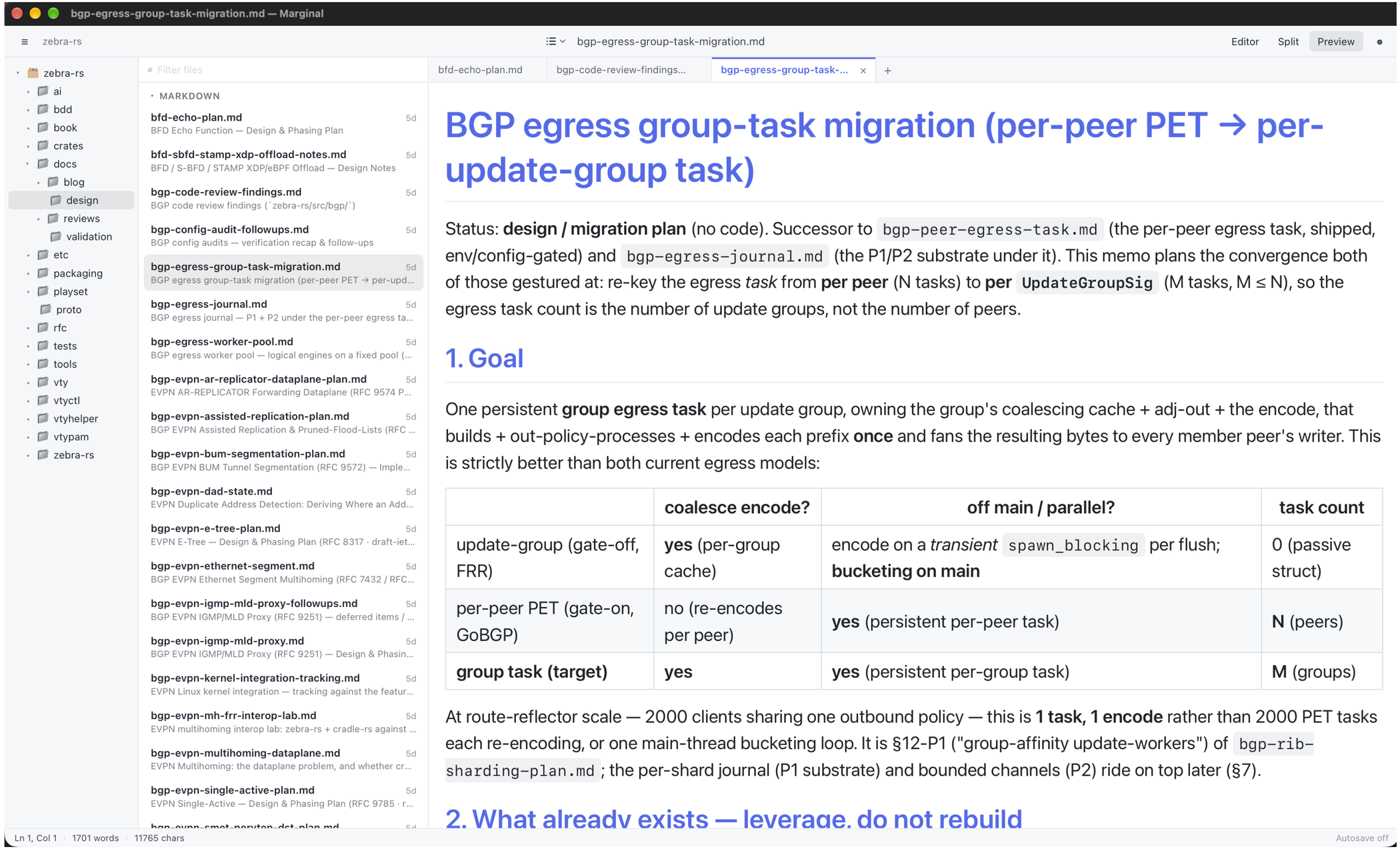
従来：年間 **約12万行**

注：AI活用の年間値は3ヶ月の実績（約60万行）を4倍した換算値

設計　実装　検証



設計 – 全て文章として証拠を残す - 数百のデザインドキュメント

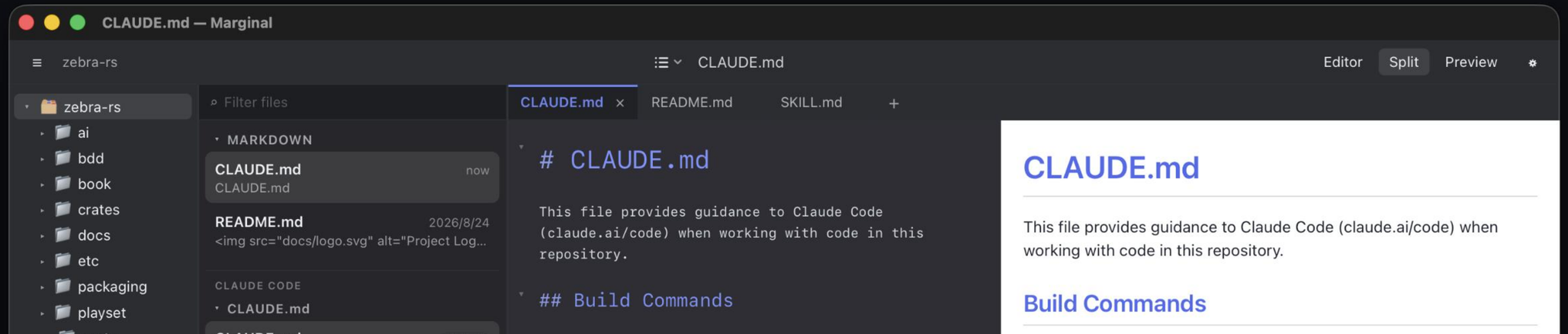


A markdown editor built for coding agents.

Marginal finds the CLAUDE.md, skills, rules, sub-agents, and commands in effect for your project and lists them beside your notes. Native on macOS, Windows, and Linux, with your files kept as plain markdown on disk.

Download Marginal | Read the manual

Version 1.0 · macOS, Windows, and Linux — 14-day free trial, no card



検証 – TDD is DEAD, BDD is the way to go

1,861のBDDシナリオが、AIが書いたコードの振る舞いを32並列15分で全件検証する

	Features	Scenarios
All .feature files	389	1,866
Disabled (bgp_lua_gbp.feature, tagged @disabled)	1	5
Enabled	**388**	**1,861**

01

仕様がそのままAIへの指示になり、失敗した場合自動修正

Given / When / Then で書いた期待動作を、AIが実装の目標として直接読み取れる

02

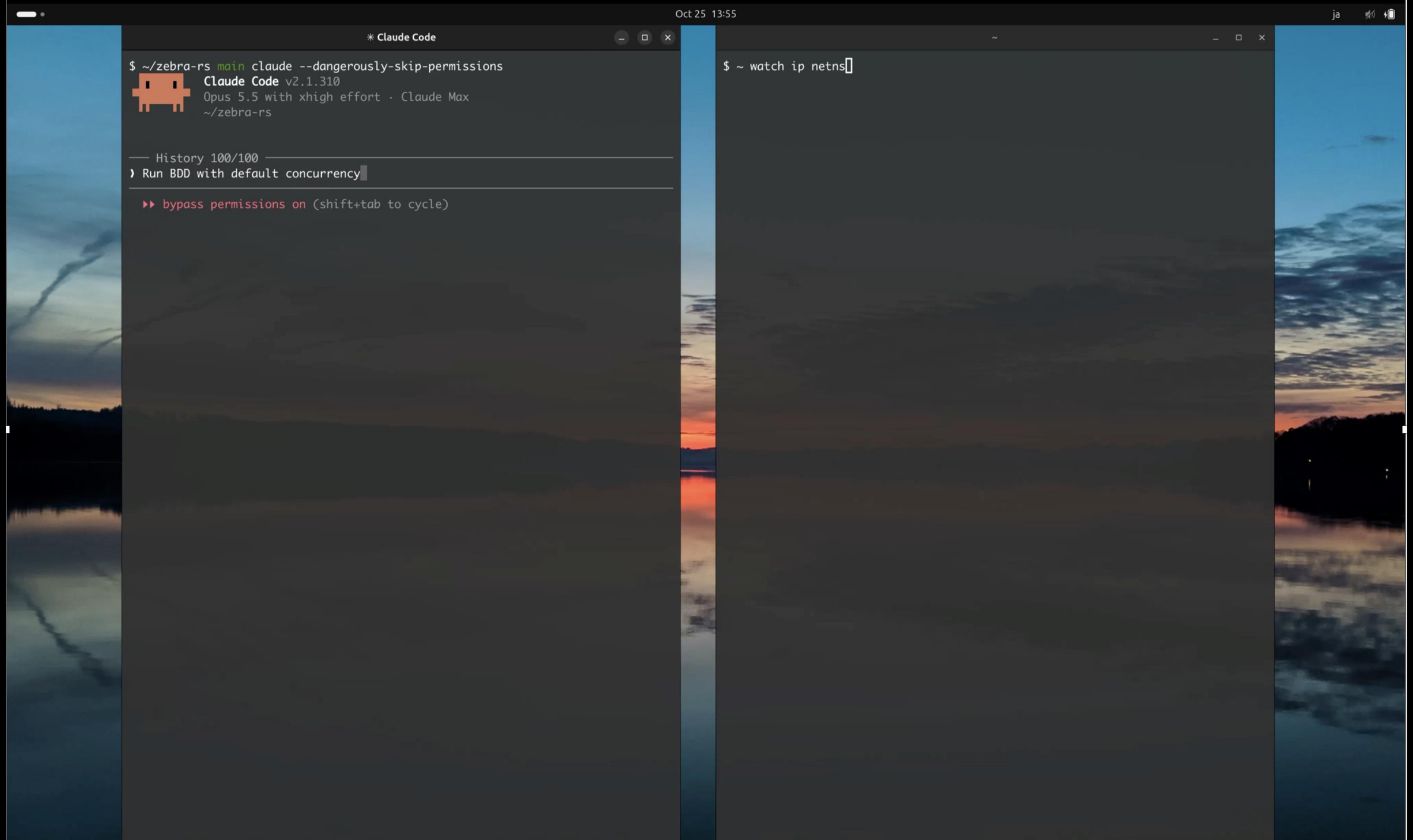
生成コードが既存の実装を壊していないかを機械的に判定できる

AIの変更が既存の振る舞いを壊していないかを、1,861シナリオで毎回確認する

03

短いフィードバックでAIの修正サイクルが回る

32並列なら6時間のBDDが、15分で全件の結果が返り、AIは1日に何度でも修正と再検証を繰り返せる


Oct 25 13:55
ja
✳ Claude Code
$ ~/zebra-rs main claude --dangerously-skip-permissions
Claude Code v2.1.310
Opus 5.5 with xhigh effort · Claude Max
~/zebra-rs
History 100/100
❯ Run BDD with default concurrency
▸▸ bypass permissions on (shift+tab to cycle)
~
$ ~ watch ip netns

Passed Allure Report

October 25, 2026 at 1:30:35 PM, with exit code 0

100%

Results 1861 | Quality Gates 0 | Global Attachments 0 | Global Errors 0

Total	Passed
1861	1861

Name or ID

More filters

Total 1861 | Passed 1861

Sort by: Order Earliest

- BGP IPv4-unicast read paths at N>1 (show / session-up sync read the empty main shard) 23
- BGP-LS carries the IS-IS TE performance metrics 15
- IS-IS TI-LFA fast-reroute over SR-MPLS 15
- IS-IS LSP fragmentation (IPv4) at lsp-mtu-size 400 and 1500 14
- BGP EVPN Ethernet Segment discovery (RFC 7432 Type-4) 12
- BGP EVPN VPWS multihoming — ESI and DF-elected P/B (RFC 8214 §5) 12
- BGP route-map match clauses 12
- OSPFv2 TI-LFA fast-reroute over SR-MPLS 11
- OSPFv3 TI-LFA fast-reroute over SR-MPLS 11
- Per-VRF BGP neighbor config-echo in the neighbor detail show 11
- BGP multipath (maximum-paths) 10
- BGP Roles and Only-to-Customer route-leak prevention (RFC 9234) 10
- IS-IS Level-1 / Level-2 interaction across an L1 area and an L2 backbone 10

AI時代のセキュリティ

Security in AI Era

Security - BGP CVEs in 20 years

CVE-2006-2276	2006	sh ip bgp CPU exhaustion
CVE-2007-1995	2007	MP_REACH/MP_UNREACH length validation
CVE-2007-4826	2007	Malformed OPEN / COMMUNITY NULL deref
CVE-2009-1572	2009	AS_PATH string exceeds buffer
CVE-2010-1674	2010	Extended-community NULL deref
CVE-2010-1675	2010	AS_PATHLIMIT attribute
CVE-2010-2948	2010	ORF record stack overflow
CVE-2010-2949	2010	Unknown AS segment type NULL deref
CVE-2011-3327	2011	ecommunity_ecom2str heap overflow
CVE-2012-0255	2012	AS4 capability / OPEN buffer assertion
CVE-2012-1820	2012	ORF capability TLV over-consume
CVE-2013-6051	2013	bgp_attr_unknown uninitialised length
CVE-2015-7361	2015	FortiOS HA - unauthenticated ZebOS shell access
CVE-2016-2342	2016	VPNv4 NLRI stack overflow
CVE-2016-4049	2016	bgp_dump_routes_func size checks
CVE-2017-15865	2017	Attribute length mishandling -> info leak
CVE-2017-16227	2017	AS_PATH size miscalculated on send
CVE-2018-5378	2018	NOTIFY data bounds on bad attr length
CVE-2018-5379	2018	Double free, cluster-list / unknown attrs
CVE-2018-5380	2018	Code-to-string table overrun
CVE-2018-5381	2018	Capability parse infinite loop
CVE-2018-13371	2018	FortiOS - regular user can change routing via ZebOS
CVE-2018-17539	2018	bgpd DoS via AS path with >=8 ASN elements
CVE-2019-5892	2019	VNC attribute 255 session flap
CVE-2021-22128	2021	FortiProxy SSL VPN - reach ZebOS shell
CVE-2022-26126	2022	isis_nb_notifications.c strdup overflow
CVE-2022-36440	2022	peek_for_as4_capability assertion
CVE-2022-37032	2022	bgp_capability_msg_parse OOB read
CVE-2022-37035	2022	UAF race, NOTIFY vs packet processing
CVE-2022-40302	2022	RFC 9072 extended-length OPEN option
CVE-2022-40318	2022	Same, bgp_open_option_parse
CVE-2022-43681	2022	OOB read at OPEN option length octet
CVE-2023-31489	2023	bgp_capability_llgr() DoS
CVE-2023-31490	2023	bgp_attr_psid_sub() DoS
CVE-2023-38406	2023	Flowspec NLRI length zero
CVE-2023-38407	2023	Labeled-unicast read past stream
CVE-2023-38802	2023	Tunnel Encapsulation attribute 23
CVE-2023-41358	2023	NLRI processed when attr length is zero
CVE-2023-41359	2023	AIGP OOB read
CVE-2023-41360	2023	ORF header read ahead-of-stream
CVE-2023-41361	2023	Oversized rcv software version
CVE-2023-41909	2023	Flowspec NULL deref
CVE-2023-45886	2023	bgpd DoS via malformed UPDATE attribute
CVE-2023-46752	2023	Malformed MP_REACH_NLRI crash
CVE-2023-46753	2023	UPDATE without mandatory attributes
CVE-2023-47234	2023	MP_UNREACH_NLRI plus trailing NLRI
CVE-2023-47235	2023	EOR not treated as withdraw
CVE-2024-31948	2024	Malformed Prefix SID attribute
CVE-2024-31949	2024	MP/GR dynamic capability infinite loop
CVE-2024-44070	2024	bgp_attr_encap TLV length
CVE-2026-5107	2026	EVPN type-2 route access control
CVE-2026-29128	2026	Satellite receiver world-readable configs
CVE-2026-37457	2026	bgp_flowspec_op_decode() off-by-one
CVE-2026-37458	2026	MP_REACH_NLRI missing input validation
CVE-2026-37459	2026	Integer underflow on crafted UPDATE
CVE-2026-37460	2026	rfapiRibBi2Ri() VNC validation

2006年から2026年の**20年間**の GNU Zebra, Quagga, FRR, ZebOSのBGP関連CVEは**合計５６件**
そのうち**90%がパケットパーサー関連**のオーバフローおよびアンダーフロー

```rust
/// Parse one attribute from the Path Attributes field.
pub(crate) fn parse(r: &mut Reader<'_>) -> Result<Self, ParseError> {
    /// ...
    let decoded = match code {
        type_code::ORIGIN => {
            let origin = Origin::from_code(value[0]).ok_or_else(|| {
                malformed(
                    ErrorKind::InvalidOriginAttribute,
                    "ORIGIN attribute has an undefined value",
                )
            })?;
            AttributeValue::Origin(origin)
        }
        type_code::AS_PATH => AttributeValue::AsPath(AsPath::parse(value)?),
        type_code::NEXT_HOP => {
            let address = Ipv4Addr::from(read_u32(value));
            if !is_unicast_host_address(address) {
                return Err(malformed(
                    ErrorKind::InvalidNextHop,
                    "NEXT_HOP attribute is not a valid IP host address",
                ));
            }
            AttributeValue::NextHop(address)
        }
        type_code::MULTI_EXIT_DISC => AttributeValue::MultiExitDisc(read_u32(value)),
        type_code::LOCAL_PREF => AttributeValue::LocalPref(read_u32(value)),
```

CLAUDE CODEEに書かせた例

switch(match)文で条件分岐

LOCAL_PREFは４バイト読み込み

典型的な実装だが。。。

こういう実装がBugの温床に

```rust
fn parse_attributes(bytes: &[u8]) -> Result<Vec<PathAttribute>> {
    let mut c = Cursor::new(bytes, protocol(3, 1, &[]));
    while !c.rest.is_empty() {
        /// ...
        let mut v = Cursor::new(value, protocol(3, 5, raw));
        let value = match type_code {
            1 => AttributeValue::Origin(match v.u8()? {
                0 => Origin::Igp,
                1 => Origin::Egp,
                2 => Origin::Incomplete,
                _ => return Err(protocol(3, 6, raw)),
            }),
            2 => AttributeValue::AsPath(parse_as_path(value)?),
            3 => {
                let address = v.ip()?;
                if !is_unicast_host(address) {
                    return Err(protocol(3, 8, raw));
                }
                AttributeValue::NextHop(address)
            }
            4 => AttributeValue::MultiExitDisc(v.u32()?),
            5 => AttributeValue::LocalPref(v.u32()?),
```

CODEXに書かせた例

Cursorを使っているのはアイデア

LOCAL_PREFで４バイト読み込みは同じ

switch文での型分岐

こういう実装がBugの温床に

では、実コードは。。。

```rust
#[derive(NomBE, Clone)]
#[nom(Selector = "AttrSelector")]
pub enum Attr {
    #[nom(Selector = "AttrSelector(AttrType::Origin, None)")]
    Origin(Origin),
    #[nom(Selector = "AttrSelector(AttrType::AsPath, Some(false))")]
    As2Path(As2Path),
    #[nom(Selector = "AttrSelector(AttrType::AsPath, Some(true))")]
    As4Path(As4Path),
    #[nom(Selector = "AttrSelector(AttrType::NextHop, None)")]
    NextHop(NexthopAttr),
    #[nom(Selector = "AttrSelector(AttrType::Med, None)")]
    Med(Med),
    #[nom(Selector = "AttrSelector(AttrType::LocalPref, None)")]
    LocalPref(LocalPref),
    /// ...
}

#[derive(Clone, NomBE, PartialEq, Eq, Hash)]
pub struct LocalPref {
    pub local_pref: u32,
}
```

実際のリリースコード

パーサーコードは全て型アノーテーション

読み込み時の長さ判断をコードに露出させない

パーサーコンビネーターによりコレクション
などにもアノーテーションで対応できる

なにより人間が書いてもAIが書いてもまったく
同じコードが期待できる

	POWER	SKILL	IDEA
	処理量・速度・持続力	言語・プロトコル・ライブラリの 知識と実装力	複数の課題を横断的に解決する能力、 全体的なトレードオフの把握力
AI	**優位** 24時間、大量のコードを 高速に並列で生成	**優位** 主要言語・RFC・ ライブラリを横断的に把握	既存パターンの 組み合わせが中心
人間	時間と体力に 限りがある	経験した分野に 限られる	**優位** 複数の課題を一気に解決する 施策の発見

AIは**POWERとSKILL**で人間を上回る。
ただし**IDEA**は、いまの時点では**人間が優位**だし、ひょっとすると今後もそうかもしれない

rustyssl

Mission

rustyssl is a **memory-safe Rust reimplementation of OpenSSL** (libcrypto + libssl) that works as a **drop-in binary replacement**. An application built against OpenSSL must run unchanged with rustyssl's libraries in its place, with no recompile and no source changes.

Why it exists: **security**. OpenSSL's history of memory-safety and parsing CVEs (Heartbleed, ASN.1 and X.509 parser bugs, the punycode overflow, and others) comes mostly from C. rustyssl removes that class of bug and keeps the exact C API/ABI, so adopting it costs users nothing.

Priority order when goals conflict:

1. **Security and correctness**
2. **ABI and behavioral compatibility** with OpenSSL
3. Performance

To try it: docs/TRY-ON-UBUNTU.md (build it, run programs on it with `LD_LIBRARY_PATH`, or install it side by side on Ubuntu 22.04 and 24.04); docs/MIGRATION.md for other distributions.

Why a Rust drop-in, and how its claims are checked (Miri, fuzzing, every OpenSSL CVE): docs/WHY-RUSTYSSL.md.

Plans: docs/IMPLEMENTATION_PLAN.md (phased steps) and docs/TEST_PLAN.md (verification strategy). Keep both current: mark steps done and record decisions as they're made.

Compatibility target

Milestone	ABI	Libraries
Primary	**OpenSSL 3.x**: 3.5 LTS symbol set first, then the 3.6 additions	`libcrypto.so.3`, `libssl.so.3` (macOS: `lib*.3.dylib`)
Later	**OpenSSL 4.0** (ENGINE API removed, ECH	`libcrypto.so.4`, `libssl.so.4` from the

OpenSSLバイナリードロップイン置換という手法

	メモリ安全	既存のバイナリがそのまま動くか
OpenSSL / LibreSSL / BoringSSL	いいえ（C）	OpenSSL は可。LibreSSL は API/ABI が異なり、BoringSSL は ABI が安定していない
rustls、Go の crypto/tls など	はい	いいえ。別の API への書き換えか、再ビルドが必要
rustyssl	**はい** FFI 層を除く	はい。同じ libcrypto.so.3 / libssl.so.3、同じシンボルで再コンパイル不要

Fully Compatible 6,502 Exports

OpenSSL 3.6.5 のエクスポートを全実装（libcrypto 5,899 + libssl 603）

unsafe は FFI 層だけ、変更のたびに Miri で検査

01 **攻撃者の入力に面するコードは安全な Rust**

ASN.1、X.509、PKCS、CMS、TLS、DTLS、QUIC、設定ファイルは #![forbid(unsafe_code)]

02 **unsafe は 1 ブロック 1 操作、すべてに // SAFETY:**

モジュールごとに unsafe 予算があり、数が増えると CI が失敗するため、自動で検証できる

03 **Miri がプルリクエストごとに FFI 層の全テストを実行**

範囲外アクセス、解放後の使用、二重解放、リーク、エイリアシング違反を確実に検知

約38,000

unsafe ブロック

約 6,500 エクスポートに対して

unsafeの使用には予算があり、

Miri が見つけた実際のバグ

RSA_up_ref が鍵をリーク

呼ばれたすべての鍵をリークしていた。プロバイダーの操作すべてに影響。Miri のリーク検査が発見

後処理が Stacked Borrows のもとで未定義動作

破棄中のオブジェクトへ戻る呼び出し時に、破棄の前に後処理を実行するようハンドルを設計し直した

108 個の fuzz ターゲットを継続的に実行

108

fuzz ターゲット

- DER/BER・X.509 の証明書と CRL
- PKCS・CMS・OCSP・TS・CMP
- PEM・base64
- TLS のメッセージとレコード・DTLS
- QUIC のパケット・フレーム・パラメータ
- 設定ファイルの言語

クラッシュだけでなく論理エラーも

モデルとの比較、ラウンドトリップ検査

PR毎にCIで実行

PR ごとに ClusterFuzzLite
毎週は全ターゲット

panic はすべてRust層で捕捉

Rust層でpanicしても、決してC の側へは伝達せず、安全側に失敗する

EVP_PKEY_CTX_set_params のリーク

レガシーな鍵メソッドの ctrl 用にコピーしたバイト列が未解放（meth ターゲットの LeakSanitizer）

TLS エンコーダーの非対称性

応答のない CertificateStatus が、デコーダーの拒否する形に再エンコードされていた（tls_msgs のラウンドトリップ検査）

OpenSSL の全 CVE 304 件を、1 件ずつ分類しテスト

235
回帰テスト
両ライブラリで実行

19
別のスイートで確認
dudect、本家のテスト

2
CLI のテストで確認

47
該当なし
SSLv2、FIPS など

1
未対応
CVE-2018-0737

Rust では防げない、論理・プロトコル・タイミングのバグもテスト

CVE-2014-0224
CCS インジェクション

CVE-2015-1793
代替チェーン（CA でない証明書）

CVE-2016-2107
CBC パディングオラクル（定数時間）

CVE-2003-0131
Bleichenbacher / Marvin

CVE-2014-3566
POODLE のフォールバック

CVE-2022-0778
BN_mod_sqrt のサービス妨害

コンパイラの最適化による障害：LLVM が CBC パディングのマスクを分岐に変えていた（dudect |t| 18〜20）→ 最適化バリアで修正

ドロップイン互換でも、安全でない動作は真似しない

呼び出し	OpenSSL の動作
OPENSSL_cleanse(NULL, n)	アドレス 0 に書き込みクラッシュ
lhash の doall	コールバックが次の要素を削除すると、解放済みのメモリをたどりクラッシュ
OPENSSL_DIR_end	ダングリングポインタが残り、2 回目の呼び出しで二重解放でクラッシュ
s_client -starttls pop3	初期化されていないメモリを表示することがある
ocsp -port	port=0指定で、CPU 100% のまま回り続ける

rustyssl

安全に失敗する

OpenSSL がクラッシュや誤動作をする箇所で、安全側に失敗する

例：OPENSSL_cleanse(NULL, n) は何もしない

各ケースはテストとともに docs/DEVIATIONS.md に記録

ファイルを置くか、イメージを置くか

方式1　ファイルをコピー

従来型　Ubuntu・Debian・Fedora・Alpine

disk



方式2　イメージをインストール

アプライアンス型 A/B　VyOS・SONiC

disk



	方式1　ファイルをコピー	方式2　イメージをインストール
インストール	約2.4万ファイルを展開	**squashfs 1つ（約260 MB）をコピー**
アップグレード	apt でその場更新。途中で失敗しうる	**イメージ全体を入れ替え。アトミック**
ロールバック	仕組みなし	**前の GRUB エントリで起動**
機器のばらつき	時間とともに増える	**全機器がテスト済みイメージで動く**
設定	**/etc のどこにでも**	永続領域 config/ に置き、移行が必要
ディスク使用量	**システム1つ分**	イメージ2つ以上

OS はリブートで、アプリはライブで更新

アプリ層 systemd-sysext　　ライブ更新

zebra-rs・cradle-rs（Rust クレートは内蔵）

extensions/*.raw

ベースイメージ A/B　　リブート

カーネル・glibc・libpam など

images/<ver>/　GRUB で切り替え

設定 Persistent領域　　永続

/etc/zebra-rs・ネットワーク・SSH

config/　イメージ更新後も残る

ライブ更新の手順

01	新しい zebra-rs-<ver>.raw を extensions/ に置く
02	systemd-sysext refresh で /usr のオーバーレイを読み直す
03	systemctl daemon-reload、zebra-rs を再起動
戻す	ロールバック時は前の .raw に戻して refresh

課題

postinst の処理をファイルに置き換える（setcap → xattr、sysusers.d など）

無停止の再起動は未検証：カーネル経路の保持、BGP GR、eBPF の bpffs ピン留め

AI時代のアルゴリズム

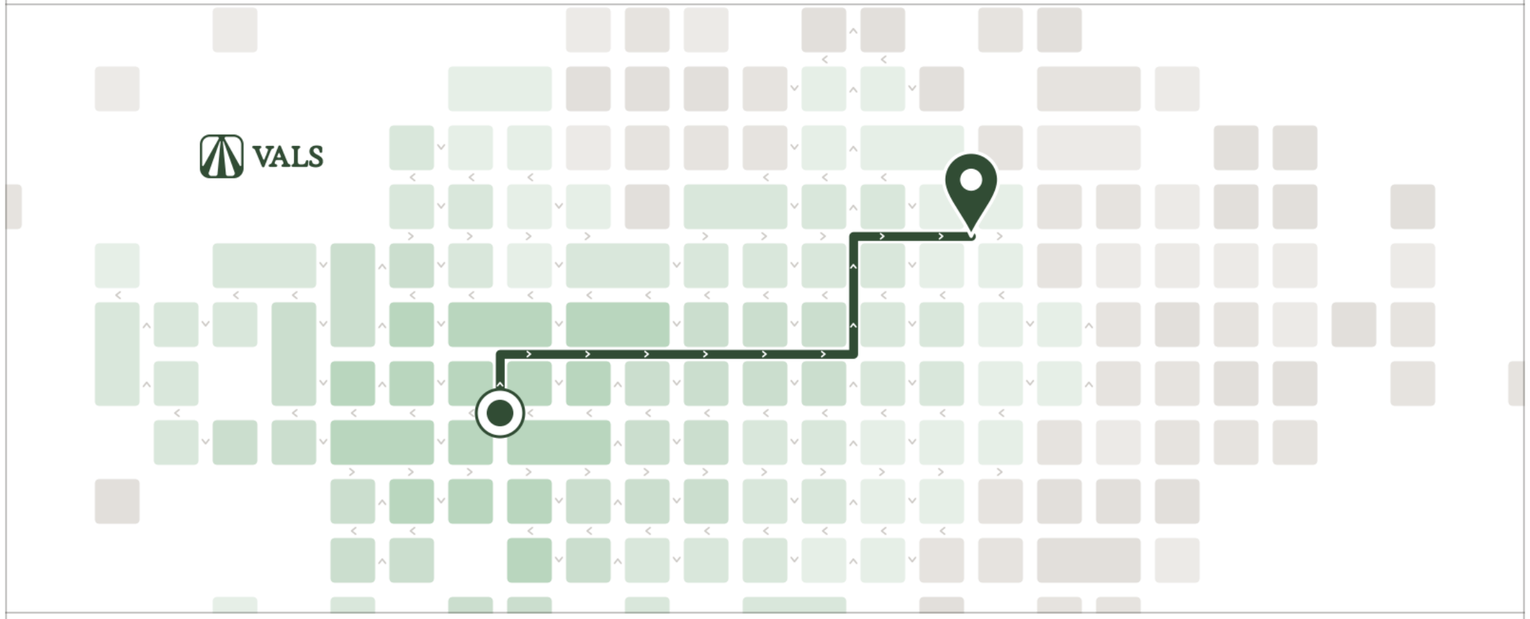
Algorithm in AI Era

BLOG RESEARCH

A Faster Shortest Path Algorithm

We asked ten Claude Opus 5.5 agents to find a faster shortest-path algorithm and prove it in Lean. Within 15 hours, they produced C-HD: a formally verified improvement over the published bounds.

GEBY JAFF • 09/20/2026



A Faster Shortest Path Algorithm
Problem
Solution
Actual Performance
Formal Verifications
How much improvement does C-HD deliver?
Elicitation



ダイクストラ法よりも高速！66年ぶりの快挙を成し遂げたアルゴリズムについて(1/2)

2025/06/29に公開 2026/03/14

19
5

アルゴリズムとデータ構造 論文解説 Tech

はじめに

Polaris.AIでお手伝いをさせていただいているSuiginと申します。筆者はアルゴリズムに興味があり、最近最短経路問題について面白い論文が出たので、そちらの解説をさせていただければなと思います。

すこしAIからは離れた内容ですが、お付き合いいただければ幸いです。

この記事の目標

この記事で扱わせていただく論文は以下になります。

H
Suigin
フォロー

目次

- はじめに
- この記事の目標
- 1. 最短経路問題 [1]
- 2. ダイクストラ法
- 3. 分割統治法の適用
- BMSSP (Bounded Multi-Source Shortest Path)
- BMSSPの実装
- BMSSPの計算量
- 今回のまとめ

AIにできること、できないこと - D.Knuth ”ART Allot"

3.3 Avoiding Recursion

Function *allot*() is one of two keys of the ART and it uses recursion. Changing the *allot*() algorithm from recursion to loop increases insertion and deletion performance. Algorithm 8 shows a non-recursive version of *allot*() with element consolidation.

Algorithm 8: Allotting route r (non-recursive)
Input: array pointer: X, smallest fringe index in X: t, base index: b, old route pointer: q, new route pointer: r
Output:
allot(X, t, b, q, r)
(1) ArrayPointer Y
(2) Int $j \leftarrow b$
(3)
(4) **if** $j \geq t$ **then**
(5) **if** ($X[b]$&1) = 1 **then**
(6) $Y \leftarrow X[b]$&(~1)
(7) **if** $Y[1] = q$ **then** $Y[1] \leftarrow r$
(8) **else if** $X[b] = q$ **then**
(9) $X[b] \leftarrow r$
(10) **endif**
(11) **return**
(12)**endif**
(13)
(14) startChange:
(15) $j \leftarrow j \ll 1$
(16) **if** $j < t$ **then goto** nonFringe
(17) /* Handle fringe indices */
(18) **while true**
(19) **if** ($X[b]$&1) = 1 **then**
(20) $Y \leftarrow X[b]$&(~1)
(21) **if** $Y[1] = q$ **then** $Y[1] \leftarrow r$
(22) **else if** $X[b] = q$ **then**
(23) $X[b] \leftarrow r$
(24) **endif**
(25) **if** (j&1) = 1 **then goto** moveUp
(26) $j \leftarrow j + 1$
(27) **endwhile**
(28)
(29) nonFringe:
(30) **if** $X[j] = q$ **then goto** startChange
(31) moveOn:
(32) **if** (j&1) = 1 **then goto** moveUp
(33) $j \leftarrow j + 1$
(34) **goto** nonFringe
(35) moveUp:
(36) $j \leftarrow j >> 1$
(37) $X[j] \leftarrow r$ /* Handle non-fringe node */
(38) **if** $j \neq b$ **then goto** moveOn

```
static inline void
rtArtAllot (subtable t, int k, routeEnt *r,
            routeEnt *s, int threshold, bool fringeCheck)
{
    register int j = k;

    assert( k < threshold );

startChange:
    j <<= 1;
    if ( j < threshold ) goto nonFringe;


    /*  change fringe nodes
     */
    while (1) {
        if ( fringeCheck && isSubtable(t[j]) ) {
            if ( subtablePtr(t[j]).down[1].ent == r ) {
                subtablePtr(t[j]).down[1].ent = s;
            }
        } else if ( t[j].ent == r ) {
            t[j].ent = s;
        }
        if ( j & 1 ) goto moveUp;
        j++;
    }

nonFringe:
    if (t[j].ent == r) goto startChange;
moveOn:
    if (j & 1) goto moveUp;
    j++;
    goto nonFringe;
moveUp:
    j >>= 1;
    t[j].ent = s;               /* change non-fringe node */
    if (j != k) goto moveOn;
}
```

経路表アルゴリズムARTの一部

再帰をループに展開するKnuth先生の必殺技

AIには見つけられないと思われる

よっぽどプログラミングが下手でないと…



Stay hungry. Stay foolish.



ISBN 0-14-003950-3