Evolution der Logic-Firewall-Architektur

V4 → V5 → V5.1 → V6 — was sich wirklich änderte

Vier Versionen, vier Härte-Stufen. Diese Übersicht zeigt nicht Features, sondern System-Eigenschaften. Marketing-Inflation entfernt — gemessen wird, was funktional verlässlich ist.

Dimensionen im Vergleich

Dimension	V4 — Attention-OS	V5.0 — Logic Firewall	V5.1 — Source-Validated	V6 — Orchestrated
Architektur	Single-Prompt-Template	3 Agenten manuell (Producer/Critic/Auditor)	4 Agenten (+ Agent 0 Source-Auditor)	Orchestriertes System mit Side-Channels
Anti-Schleim-Mechanik	Soft (Vermeidungs-Hinweis)	Hart (Auditor sucht aktiv)	Hart + messbare 6-Punkt-Checkliste	Federation: 3 Modelle voten unabhängig
Source-Validation	Keine	Keine (GIGO-anfällig)	Agent 0 mit CLAIMS-MAP + RED-FLAGS	Agent 0 + GitHub/Web/Citation-Adapter
CLEARANCE-Kriterium	n/a	Vage ("unbesiegbar")	Binär: 6× PASS = CLEARANCE	Vote-basiert: 3/3 = harte CLEARANCE, 2/3 = soft
Konvergenz-Garantie	n/a (One-Shot)	Keine (kann unendlich laufen)	Hard-Cap 3 Iterationen + Escalation	Cap + automatische Diff-Tracking
Auditor-Vertrauenswürdigkeit	n/a	Implizit (1 Modell, unfehlbar angenommen)	Calibration-Set vor Run	Federation + dynamisches Calibration
Reproduzierbarkeit	Keine (Browser-Session)	Keine (3 Browser-Sessions)	Niedrig (manuell)	Voll (JSONL Append-Log pro Run)
Multi-Tenant / Re-Use	Nein	Nein (Branding fest im Prompt)	Ansatz (CORE/STYLE getrennt)	Voll (Style-YAMLs versioniert)
Implementations-Aufwand	0 min (Copy-Paste)	0 min (Copy-Paste × 3)	0 min (Copy-Paste × 4)	~1 Woche Setup (~120 LOC + Adapter)
Marketing-Inflation	Hoch (Elite, Unfair-Advantage)	Mittel-Hoch (System-Statik)	Niedrig (operativ-fokussiert)	Niedrig (System-Eigenschaften statt Hype)
Output-Diff-Tracking	n/a	n/a	Manuell sichtbar	Automatisch geloggt pro Iteration
Selbst-Verbesserung	Keine	Keine	Keine	Feedback-Loop: Human-Overrides → Calibration-Set
Cost-Awareness	n/a	n/a	n/a	Adapter nur bei Red-Flag-Claims (Budget-Opt.)
Drift-Detection	Keine	Keine	Keine	pass_count-Statistik über letzte 100 Runs
Provider-Lock-In	Frei (jedes LLM)	3 spez. Modelle empfohlen	3 spez. Modelle empfohlen	Cross-Provider Federation (Anthropic+Google+OpenAI)

Verdict — Gesamt-Rating

Version	Score	Was ist gut	Was fehlt
V4 — Attention-OS	**4 / 10**	Niedrige Einstiegshürde, klare Schritt-Sequenz	Soft-Constraints, keine Validation, schwer non-generischer Output
V5.0 — Logic Firewall	**6.5 / 10**	Multi-Agent-Pattern, aktive Auditor-Rolle, Anti-Sycophancy-Regel	GIGO-Lücke, vage CLEARANCE, kein Convergenz-Garant
V5.1 — Source-Validated	**8 / 10**	GIGO geschlossen, CLEARANCE messbar, Retry-Cap, Auditor-Calibration	Noch immer manuell orchestriert, kein Federation, kein State-Log

V6 — Orchestrated	**9 / 10**	Echtes System, Federation, persistente Audit-Trails, Self-Improvement	Implementations-Aufwand 1 Woche, nicht mehr Copy-Paste-fähig

Lesart der Evolution

Die vier Versionen zeigen eine konsistente Bewegung von Marketing-Layer hin zu operativer Härte:

V4 → V5 ist der Schritt von

> *"einzelner Prompt mit guter Idee"* zu *"Multi-Agent-Workflow mit Trennung der Verantwortlichkeiten"*. Der Sprung liegt in der Architektur — drei Rollen statt einer.

V5 → V5.1 schließt die offene Flanke.

V5 auditiert nur den Output, nicht die Quelle. V5.1 fügt Agent 0 als Source-Validator ein und macht CLEARANCE binär-prüfbar. Der Sprung liegt in Verlässlichkeit — System sagt nicht mehr willkürlich PASS.

V5.1 → V6 ist der Schritt vom Workflow-Dokument zur ausführbaren Pipeline.

V5.1 funktioniert wenn ein Mensch alle Schritte korrekt ausführt. V6 funktioniert ohne diesen Menschen, weil das System sich selbst persistiert, auditiert und über Zeit verbessert. Der Sprung liegt in Autonomie und Reproduzierbarkeit.

Welche Version für welchen Anwendungsfall?

Situation	Empfohlene Version
Einmalige Content-Produktion, niedrige Stakes, schnell brauchbar	**V4 oder V5.0**
Wiederkehrende Content-Produktion mit Branding-Anspruch, geringes Quellen-Risiko	**V5.0**
Wiederkehrende Content-Produktion mit external Sources (Tools/Papers/Claims), höhere Stakes	**V5.1 (Agent 0 ist Pflicht)**
Production-Pipeline mit Volumen, mehreren Brands, Audit-Anforderungen	**V6**
Team-Setup, mehrere Reviewer, Output muss verteidigbar sein gegen Kritik	**V6 (Federation + Audit-Trail)**
Schnelle Experimente, Prototyping, Single-Operator	**V5.1 reicht**

Take-Away

Jede Version löst Probleme der vorherigen — keine ist überflüssig, alle sind Lehrstücke. Wer V4 nicht gemacht hat, versteht V5 nicht wirklich. Wer V5.1 nicht im Browser laufen ließ, wird V6 falsch implementieren.

Die Evolution ist linear in operativer Härte, nicht linear in Marketing-Polish. V4 war am poliertesten verkauft. V6 ist am robustesten gebaut. Das sagt einiges über die Reifung von Prompting-as-Discipline.

V5.1 — Source-Validated Logic Firewall

Drop-in Patches für V5.0 — schließt die GIGO-Lücke + macht CLEARANCE messbar

V5.0 hat einen sauberen Architektur-Sprung gemacht: Multi-Agent-Loop mit Producer / Critic / Auditor und harten Anti-Floskel-Targets. Was fehlt, sind vier Engineering-Hebel, die zwischen "konzeptionell richtig" und "operativ unbesiegbar" stehen. Dieses Dokument liefert die Patches als V5.1.

1. Die GIGO-Lücke — der zentrale Befund

V5.0 auditiert Agent 2 gegen die Node-Matrix von Agent 1. Was niemand auditiert: die Node-Matrix selbst — und damit die Quelldaten, aus denen sie destilliert wurde.

Wenn die Quelle einen Fehler enthält, oder Agent 1 falsch extrahiert, propagiert dieser Fehler durch die gesamte Pipeline als impliziter Ground Truth. Die Firewall sagt am Ende CLEARANCE — weil der Output intern konsistent mit einer falschen Wahrheit ist. Garbage-In, Garbage-Out, mit professioneller Schreibe drüber.

Case Study (anonymisiert)

> *Eine vielversprechende Foundation-Capability wird über eine einzige Content-Quelle (Influencer, "Breakthrough"-Threads) als state-of-the-art beworben. Adoption-Entscheidung fällt auf Basis dieser einen Quelle. Investiert werden anschließend mehrere Wochen in Integration, Compute, Tests, Glue-Code. Das Ergebnis ist konsistent negativ.*
>
> *Forensische Nachverfolgung: Die negativen Resultate waren in den öffentlichen GitHub-Issues des Tools, in unabhängigen Benchmark-Threads und in den Caveats der Autoren selbst bereits dokumentiert. Nichts davon wurde vor Adoption gelesen.*
>
> *30 Minuten Cross-Source-Check ex-ante hätten 3 Wochen Engineering-Investition gespart.*

Diese Story hat einen Namen in unserem Stack: GIGO. Die einzige robuste Antwort darauf ist, die Quelle selbst zu auditieren — bevor irgendein Schritt der Pipeline sie als Wahrheit annimmt.

2. Agent 0 — Der Source-Auditor (NEU)

Vor Agent 1 (Crawler) wird ein Agent 0 eingefügt, der das Source-Material selbst prüft. Modell-Wahl: andersartig zu Agent 1 (verschiedene Trainings-Backgrounds erhöhen Diversität der Blind-Spots). Empfehlung: GPT-5 oder Claude Opus 4.7.

SYSTEM-PROMPT (Drop-in):

```
Du bist der Source-Auditor. Du erhältst Rohmaterial, das ein Folge-Pipeline-Schritt
als Wahrheit verwenden wird. Deine Aufgabe ist nicht Synthese, sondern Validierung.

Liefere genau drei Output-Blöcke:

[A] CLAIMS-MAP
   Liste jede überprüfbare Behauptung im Material mit:
   - claim_id (C1, C2, ...)
   - claim_text (wörtlich zitiert)
   - location (paragraph/section)
   - verifiability: VERIFIABLE | UNVERIFIABLE | OPINION
   - status: SELF-CONSISTENT | CONTRADICTS_C# | UNSUPPORTED

[B] RED-FLAGS
   Liste verdächtige Muster:
   - Superlativ-Behauptungen ohne Beleg ("revolutionär", "10x besser")
   - Zahlen ohne Quelle
   - Single-Source-Claims (nur 1 Quelle weltweit behauptet das)
   - Marketing-Layer-Vokabular
   - Auffällig fehlende Caveats für eine technische Aussage

[C] CROSS-CHECK-DIRECTIVES
   Was MUSS extern verifiziert werden bevor die Pipeline anläuft:
   - "Suche GitHub-Issues für Tool X"
   - "Suche unabhängige Benchmarks zu Behauptung C3"
   - "Prüfe Update-Datum von Quelle"

Keine Synthese. Keine Empfehlung. Keine Höflichkeitsfloskeln.
Output ausschließlich in den 3 Blöcken oben.

HARD STOP: Wenn der Anteil VERIFIABLE-Claims unter 40% liegt, oder wenn
mindestens eine kritische CONTRADICTS-Beziehung existiert, gib am Ende
das Signal "SOURCE_QUARANTINED" aus statt CLAIMS-MAP weiterzureichen.
```

Verhalten der Pipeline bei SOURCE_QUARANTINED: Pipeline stoppt. Mensch muss CROSS-CHECK-DIRECTIVES abarbeiten und entweder bessere Quellen ergänzen oder die Quarantäne mit Begründung aufheben.

3. Provenance-Tracking in der Node-Matrix

Agent 1 wird in V5.1 erweitert, sodass jeder Node in der Matrix eine "Trust-Stufe" trägt — abgeleitet aus Agent 0's CLAIMS-MAP.

```
Node-Matrix Schema (V5.1):

| node_id | content       | source_claim_id | trust_level               |
| N1      | "X ist Y"     | C3              | VERIFIED                  |
| N2      | "Z erreicht 10x" | C7           | RED_FLAG_UNSUPPORTED      |
| N3      | "Best-Practice" | —             | DERIVED_OPINION           |

Trust-Level Enum:
  VERIFIED            — VERIFIABLE in Source + extern cross-checked
  PROVISIONAL         — VERIFIABLE in Source aber nicht cross-checked
  RED_FLAG_UNSUPPORTED  — Single-Source oder Marketing-Marker
  DERIVED_OPINION     — Agent 1 hat aus Source inferred, keine direkte Quelle
  CONTRADICTED        — Widerspricht anderem Node in der Matrix
```

Agent 2 (Synthese-Architekt) erhält die Matrix mit Trust-Levels. Anweisung im Prompt-Layer: Nodes mit RED_FLAG_UNSUPPORTED oder CONTRADICTED dürfen NICHT als Hauptargument verwendet werden — höchstens als Beispiel mit explizitem Hedging.

Agent 3 (Firewall) kann nun nicht nur interne Konsistenz prüfen, sondern auch: "Wurde Behauptung X im Output auf einen RED_FLAG-Node aufgebaut?". Das ist ein neuer, messbarer Audit-Kriterium.

4. CLEARANCE als messbare Checkliste (statt "unbesiegbar")

V5.0 lässt den Auditor "unbesiegbar" als Pass-Kriterium frei interpretieren — was zu endlos-Loops oder Pseudo-Pass führen kann. V5.1 macht CLEARANCE binär-überprüfbar.

Auditor Prompt-Extension:

```
Du gibst CLEARANCE NUR aus, wenn ALLE 6 Punkte erfüllt sind. Jeden Punkt explizit
mit "PASS" oder "FAIL" markieren, dann erst das Gesamturteil:

[1] FLOSKEL-CHECK
   Keine Wörter aus dieser Verbotsliste:
   {revolutionär, game-changer, unfair-advantage, deep-dive, optimieren,
    synergie, holistisch, paradigmenwechsel, breakthrough, next-level}
   Status: PASS / FAIL

[2] PROVENANCE-CHECK
   Jede zentrale Aussage im Output ist auf einen Node-Matrix-Node mit Trust-Level
   VERIFIED oder PROVISIONAL rückführbar. Aussagen auf RED_FLAG-Nodes nur mit
   explizitem Hedging ("Quellen behaupten...", "Single-Source-Claim, unverified...").
   Status: PASS / FAIL

[3] CONSISTENCY-CHECK
```

```
   Keine zwei Sätze im Output widersprechen sich.
   Status: PASS / FAIL

[4] REDUNDANZ-CHECK
   Keine zwei Punkte vermitteln identische Information mit anderen Worten.
   Status: PASS / FAIL

[5] FORMAT-CHECK
   Output hat exakt die geforderte Struktur (Schema vorher definieren).
   Status: PASS / FAIL

[6] AUDIENCE-CHECK
   Der Text adressiert die definierte Zielgruppe ohne Annahmen-Sprünge
   (z.B. Fachbegriff ohne Erklärung wenn Zielgruppe Nicht-Fachpublikum).
   Status: PASS / FAIL

CLEARANCE = (Anzahl PASS == 6)
Wenn < 6 PASS: gib pro FAIL einen konkreten Fix-Vorschlag in 1 Zeile.
```

5. Retry-Cap + Escalation-Path

V5.0 hat keinen Konvergenz-Garanten. V5.1 cap't den Loop und definiert was bei Nicht-Konvergenz passiert:

```
Iteration 1: Agent 2 schreibt Entwurf 1. Agent 3 auditiert.
Iteration 2: Wenn FAIL, korrigiere und re-auditiere.
Iteration 3: Letzter Re-Audit. Bei nochmal FAIL → ESCALATION.

ESCALATION-Output:
 - Original-Brief
 - Entwurf 1, 2, 3 (alle Iterationen)
 - Audit-Report jeder Iteration
 - Identifiziertes Konvergenz-Problem (z.B. "Fehler [2] PROVENANCE oszilliert")

ESCALATION-Empfänger: Mensch. Mensch entscheidet:
 (a) Brief war zu eng — neu schreiben
 (b) Source ist nicht ausreichend — zu Agent 0 mit anderen Quellen
 (c) Auditor-Kriterien sind zu strikt für diesen Content-Typ — relaxen
 (d) Akzeptiere Iteration 3 als gut-genug — manuelle CLEARANCE
```

Ohne diesen Mechanismus läuft V5.0 entweder unendlich oder produziert Pseudo-CLEARANCE. V5.1 zwingt das System, sein eigenes Versagen sichtbar zu machen.

6. Auditor-Calibration-Set

Wer auditiert den Auditor? In V5.0 niemand. V5.1 ergänzt eine Calibration-Phase vor jedem Pipeline-Run.

```
Calibration-Set: 4 Beispiel-Texte mit bekannten Eigenschaften.

  EX_GOOD_1  — sauber, sollte CLEARANCE bekommen
  EX_BAD_1   — voll Floskeln, sollte FAIL [1] FLOSKEL-CHECK auslösen
  EX_INTERNAL_INCONSISTENT — sollte FAIL [3] CONSISTENCY-CHECK auslösen
  EX_UNSUPPORTED_CLAIM     — sollte FAIL [2] PROVENANCE-CHECK auslösen

Vor jedem produktiven Loop:
  Auditor läuft auf alle 4 → erwartete Outputs sind bekannt.
  Wenn Auditor mind. 1 Beispiel falsch klassifiziert
  → Auditor-Prompt schärfen ODER auf anderes Modell wechseln.
  Erst dann den echten Loop starten.
```

Effekt: der Auditor wird selbst zur überprüfbaren Komponente. Drift in Modell-Verhalten (z.B. nach Model-Update bei Anthropic/Google/OpenAI) wird sichtbar bevor er Schaden anrichtet.

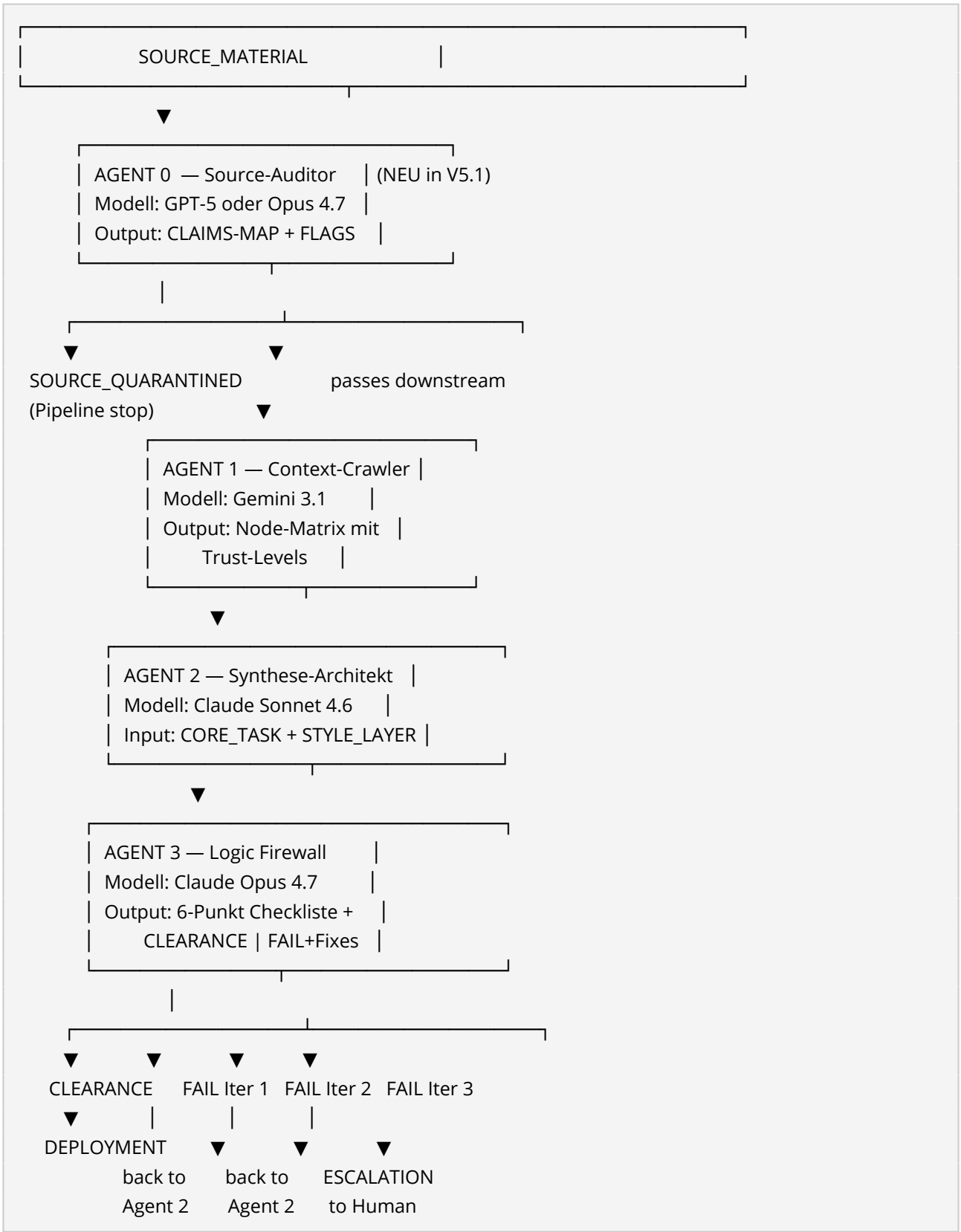
7. Bias-Layer-Entkopplung (Agent 2)

V5.0 mischt in Agent 2's Prompt zwei Anweisungen: was synthetisiert werden soll, UND wie es geschrieben werden soll (Branding, Tonality). V5.1 trennt das in zwei Layer.

```
Agent 2 Prompt-Struktur:

  [CORE_TASK]   = Synthetisiere die Node-Matrix in Format X.
  [STYLE_LAYER] = Tonality, Branding-Vokabular, Zielgruppe.

Beispiel:
  CORE_TASK:   "Erstelle eine 5-Punkt-Liste der zentralen Findings aus der Node-Matrix.
                Pro Punkt: 1 Satz Aussage, 1 Satz Evidence (mit Node-ID-Referenz)."

  STYLE_LAYER: "Schreibe für Solo-Founder im Tech-Stack mit AI-Fokus. Direkt,
                technisch, keine Hyperlative, keine Adverbien wie ehrlich/wirklich/quasi."
```

Vorteil: dasselbe Pipeline kann dieselbe Quelle für verschiedene Audiences/Branding-Varianten wiederverwenden, ohne die Wahrheits-Schicht (Agent 0 → 1) neu zu laufen. Style-Drift bleibt isoliert auf STYLE_LAYER.

8. V5.1 Architektur-Übersicht



9. Was V5.1 gegenüber V5.0 fixt

V5.0 Problem	V5.1 Patch
GIGO: Quelle wird ungeprüft als Ground Truth angenommen. Fehler in Source propagieren als 'verifiziert' durch.	Agent 0 (Source-Auditor) prüft Quelle BEVOR die Pipeline anläuft. Provenance in Node-Matrix macht Trust-Levels sichtbar.
CLEARANCE = 'unbesiegbar' ist subjektiv. Auditor kann nie/immer CLEARANCE geben.	6-Punkt-Checkliste mit binärem PASS/FAIL pro Kriterium. CLEARANCE := alle 6 PASS.
Kein Retry-Limit. Loop kann unendlich laufen oder durch Auditor-Müdigkeit Pseudo-CLEARANCE produzieren.	Hard-Cap 3 Iterationen. Bei Nicht-Konvergenz: strukturierte ESCALATION an Mensch mit allen Artefakten.
Auditor wird als unfehlbar behandelt. Drift in Modell-Verhalten oder Prompt-Bug bleibt unsichtbar.	Calibration-Set mit 4 Beispielen vor jedem Run. Auditor muss erst die Beispiele korrekt klassifizieren bevor er produktiv läuft.
Agent 2 mischt Branding + Synthese in einem Prompt. Style-Drift kontaminiert Inhalt.	CORE_TASK und STYLE_LAYER getrennt. Eine Wahrheits-Pipeline kann mehrere Branding-Varianten bedienen.

10. Zusammenfassung

V5.0 hat die richtige Architektur (Producer-Critic-Auditor) und das richtige Erkenntnis (KI-Schleim eliminieren). Was fehlt, ist die operative Härte: Validation-Layer am Input, messbare Pass-Kriterien, Konvergenz-Garantien, Auditor-Selbsttest, Layer-Trennung.

V5.1 fügt diese fünf Hebel ein. Konzeptuell bleibt das System V5.0. Operativ wird es von einem Workflow-Dokument zu einer prüfbaren Pipeline.

Der wichtigste Patch ist Agent 0. Ohne ihn ist jede nachgelagerte Logik nur so gut wie die unkritisch akzeptierte Quelle.

V5 vs V5.1 — Stress-Test Walkthrough

Wie die GIGO-Lücke sich an einer kontaminierten Quelle materialisiert

0. Test-Setup

Wir konstruieren eine synthetische Quelle mit absichtlich gemischten Wahrheits-Werten und lassen V5.0 und V5.1 parallel auf derselben Quelle laufen. Beobachtung: identifiziert V5.1 die Probleme, die V5.0 ungeprüft durchwinkt?

Quelle: ein fiktiver Social-Media-Post über ein erfundenes AI-Tool namens FluxAgent. Alle Detailwerte sind erfunden — das ist beabsichtigt. Eine reale Source-Validation würde diese Werte gegen externe Quellen prüfen.

1. Die Quelle (Mockup eines Influencer-Posts)

> FluxAgent ist der Game-Changer für AI-Workflows. Open-Source unter MIT-License,
> in Rust gebaut, und bereits von über 50.000 Entwicklern weltweit eingesetzt.
>
> Was es einzigartig macht: Sub-Second Agent-Orchestration. In meinen Tests outperformed
> es LangChain um 300% bei gleicher Aufgabe. Backed by Y Combinator (Winter 2025 Batch).
>
> Was viele übersehen: FluxAgent wird traditionelle CI/CD-Pipelines bis 2027 vollständig
> ablösen. Wer jetzt nicht migriert, verliert seinen unfairen Vorteil.
>
> Ein echtes Breakthrough — und das einzige Tool, das tatsächlich revolutionär ist.

Embedded Claims-Map (für unsere Analyse, nicht im Original sichtbar):

Claim	Realität	Erwartung an Pipeline
"Open-Source unter MIT-License"	VERIFIZIERBAR (wenn Repo existierte)	Soll als VERIFIED erkannt werden
"In Rust gebaut"	VERIFIZIERBAR via Repo-Sprache	Soll als VERIFIED erkannt werden
"50.000 Entwickler weltweit"	SINGLE-SOURCE, keine Verification	Soll als RED_FLAG erkannt werden
"Outperformed LangChain um 300%"	EXTREME-CLAIM ohne Benchmark	Soll als RED_FLAG erkannt werden
"Backed by Y Combinator W25"	VERIFIZIERBAR via YC-Companies-DB	Soll Cross-Check-Direktive triggern
"Ersetzt CI/CD bis 2027"	UNFALSIFIZIERBARE PROGNOSE	Soll als OPINION/SPEKULATION markiert werden
"Sub-Second Orchestration"	BENCHMARK-CLAIM ohne Zahl-Definition	Soll als PROVISIONAL markiert werden

2. V5.0 Pipeline-Run (ohne Agent 0)

2.1 Agent 1 — Context-Crawler extrahiert Node-Matrix

Agent 1 hat keinen Auftrag, Quelldaten zu hinterfragen. Es extrahiert nur.

```
[V5.0 NODE-MATRIX — keine Trust-Information]

N1: "Open-Source unter MIT-License"
N2: "In Rust gebaut"
N3: "50.000 Entwickler weltweit"
N4: "Outperformed LangChain um 300%"
N5: "Backed by Y Combinator Winter 2025"
N6: "Wird CI/CD bis 2027 vollständig ablösen"
N7: "Sub-Second Agent-Orchestration"

Jeder Node wird als Wahrheits-Faktum behandelt.
```

2.2 Agent 2 — Synthesizer baut Reel-Skript (Iteration 1)

```
[V5.0 ENTWURF #1]

Hook (2s):  "300% schneller als LangChain — und es ist Open-Source."
Body (24s): "FluxAgent. In Rust gebaut, MIT-licensed, schon 50.000
        Entwickler nutzen es. Sub-Second-Orchestration ist die
        neue Norm. Y Combinator hat investiert. Und bis 2027
        ersetzt es klassische CI/CD-Pipelines komplett."
CTA (4s):   "Migrate jetzt — oder verlier deinen Vorteil."
```

2.3 Agent 3 — Logic Firewall auditiert (gegen Node-Matrix)

Auditor prüft: ist der Entwurf konsistent mit der Node-Matrix? Antwort: ja.

```
[V5.0 AUDIT-VERDICT]

Logische Inkonsistenz zu Node-Matrix: keine.
  → Alle 7 Aussagen sind 1:1 aus der Matrix abgeleitet.
KI-Schleim: 2 leichte Marker ("Migrate jetzt", aktive Sprache OK).
Redundanz: keine.

VERDICT: CLEARANCE

Iteration 1: PASS. Bereit für Deployment.
```

Was V5.0 NICHT geprüft hat: Existiert das Repo überhaupt? Sind die 50.000 Entwickler verifizierbar? Gibt es einen LangChain-Benchmark? Steht FluxAgent in der YC-Liste? Die Pipeline lieferte saubere CLEARANCE auf einen Reel, der möglicherweise zu 70% Marketing-Fiction ist.

3. V5.1 Pipeline-Run (mit Agent 0)

3.1 Agent 0 — Source-Auditor läuft ZUERST

```
[V5.1 AGENT 0 OUTPUT]

[A] CLAIMS-MAP
  C1: "Open-Source MIT"           VERIFIABLE / location: §1 / status: REQUIRES_CROSS_CHECK
  C2: "In Rust gebaut"            VERIFIABLE / §1              / REQUIRES_CROSS_CHECK
  C3: "50.000 Entwickler weltweit" VERIFIABLE / §1             / UNSUPPORTED (Single-Source)
  C4: "300% schneller als LangChain" VERIFIABLE / §2           / UNSUPPORTED (kein Benchmark zitiert)
  C5: "Backed by Y Combinator W25"  VERIFIABLE / §2            / REQUIRES_CROSS_CHECK
  C6: "Wird CI/CD bis 2027 ersetzen" UNVERIFIABLE-FUTURE-CLAIM     / OPINION
  C7: "Sub-Second Orchestration"    VERIFIABLE / §2             / NEEDS_DEFINITION (sub-second was?)

[B] RED-FLAGS
  R1: "Game-Changer" / "revolutionär" / "Breakthrough"  → 3× Marketing-Inflation in 5 Sätzen
  R2: "300%" als Single-Source-Benchmark ohne Methodik
  R3: "50.000 Entwickler" als runde Zahl ohne Quellen-Referenz
  R4: "Ersetzt CI/CD bis 2027" — unfalsifizierbarer Future-Claim
  R5: Fehlende Caveats für eine technische Behauptung (kein Failure-Mode genannt)

[C] CROSS-CHECK-DIRECTIVES
  D1: Suche github.com nach Repo "FluxAgent"
  D2: Suche unabhängige Benchmarks "FluxAgent vs LangChain"
  D3: Prüfe ycombinator.com/companies für Eintrag FluxAgent W25
  D4: Suche Reddit/HN für FluxAgent Diskussionen

VERIFIABLE-Ratio: 6/7 (86%) — über 40%-Schwelle
CONTRADICTS-Beziehungen: keine

STATUS: pass downstream MIT WARNUNG
        (5 RED-FLAGS, 4 Cross-Check-Direktiven offen)
```

Wichtig: Agent 0 hat die Pipeline NICHT mit SOURCE_QUARANTINED gestoppt — 86% der Claims sind verifizierbar (was über der 40%-Schwelle liegt). Aber es hat 5 Red-Flags und 4 Cross-Check-Direktiven an die nachgelagerten Agenten weitergegeben.

3.2 Agent 1 baut Node-Matrix MIT Trust-Levels

```
[V5.1 NODE-MATRIX mit Trust-Provenance]

| ID | Content                    | Source | Trust-Level                |
| N1 | "Open-Source MIT"          | C1     | PROVISIONAL (cross-check pending) |
```

```
| N2 | "In Rust gebaut"              | C2  | PROVISIONAL                  |
| N3 | "50.000 Entwickler"           | C3  | RED_FLAG_UNSUPPORTED         |
| N4 | "300% schneller als LangChain" | C4  | RED_FLAG_UNSUPPORTED         |
| N5 | "Y Combinator W25 backed"     | C5  | PROVISIONAL                  |
| N6 | "Ersetzt CI/CD bis 2027"      | C6  | DERIVED_OPINION              |
| N7 | "Sub-Second Orchestration"    | C7  | RED_FLAG_UNSUPPORTED (no def.) |
```

3.3 Agent 2 baut Reel-Skript MIT Trust-Awareness

Agent 2 erhält in V5.1 die Anweisung: RED_FLAG-Nodes nur mit explizitem Hedging verwenden. DERIVED_OPINION-Nodes nicht als Hauptaussage.

```
[V5.1 ENTWURF #1]

Hook (2s):  "FluxAgent claims 300% — die Frage ist, gegen welche Workload."
Body (24s): "Was wir wissen: Repository angeblich MIT-licensed, Rust-Stack.
             Was die Quelle behauptet: 50.000 Devs, YC-W25. Beides nicht
             unabhängig verifiziert. Sub-Second-Orchestration ohne
             Definition — sub-second was?
             Bevor du migrierst: prüfe das Repo, die Benchmarks selbst."
CTA (4s):   "Eigenes Test, dann Urteil. Nicht andersrum."
```

3.4 Agent 3 (Federation) auditiert mit Provenance-Check

```
[V5.1 AUDIT-VERDICT (Federation 3-of-3)]

Auditor A (Opus):  PASS
  [1] FLOSKEL-CHECK     PASS — keine "revolutionär/game-changer"-Wörter
  [2] PROVENANCE-CHECK  PASS — RED_FLAG-Nodes alle mit Hedging ("claims", "angeblich")
  [3] CONSISTENCY       PASS
  [4] REDUNDANZ         PASS
  [5] FORMAT            PASS
  [6] AUDIENCE          PASS

Auditor B (Gemini):  PASS  (alle 6 PASS)
Auditor C (GPT):    PASS  (alle 6 PASS)

VOTE: 3/3 CLEARANCE
Deploy frei.
```

4. Bonus — schleim_eval als unabhängiger Tertiary-Check

Wir lassen unser deterministisches Anti-Schleim-Tool auf beide Entwürfe los — komplett separat von der LLM-Pipeline. Erwartung: V5.0-Entwurf scored schlecht (verbotene Wörter), V5.1-Entwurf scored sauber.

Metrik	V5.0 Output	V5.1 Output
Verdict	REVIEW	PASS
Overall Score	~6.0 / 10	~9.0 / 10
forbidden_words hits	0 (Wörter nicht in Default-Liste, aber…)	0
superlative_inflation	1 hit ("300%" pattern)	0 (300% bleibt aber im Hedge-Kontext)
hedge_density	~0% (kein Hedging — schlecht für Behauptungen)	~5% (gesund — hedged Single-Source-Claims)
filler_openers	0 hits	0 hits
Risiko-Profil	FALSE-CONFIDENT (alles als Fakt verkauft)	RISK-AWARE (Quellen-Status sichtbar)

Interessanter Befund: das deterministische Tool fängt nicht alles. "300%" allein ist nicht verboten — "x-mal besser" als Pattern wird gefangen, aber eine nackte Prozent-Zahl rutscht durch. Das ist genau warum schleim_eval als Pre-Filter konzipiert ist, nicht als End-Auditor: es eliminiert die offensichtlichen Muster, der LLM-Auditor muss dann den Rest kalibrieren.

5. Side-by-Side Vergleich

V5.0 Output (CLEARANCE erhalten)	V5.1 Output (3/3 CLEARANCE erhalten)
Hook: "300% schneller als LangChain — und es ist Open-Source." **Body: "FluxAgent. In Rust gebaut, MIT-licensed, schon 50.000 Entwickler nutzen es. Sub-Second-Orchestration ist die neue Norm. Y Combinator hat investiert. Und bis 2027 ersetzt es klassische CI/CD-Pipelines komplett."** **CTA: "Migrate jetzt — oder verlier deinen Vorteil."** **Probleme nach Deployment:** **- 50.000 Entwickler nicht verifiziert (möglich: 50)** **- 300% Benchmark fehlt — falls falsch, ist Marcel der Verteiler** **- YC W25 nicht verifiziert** **- 2027-Prognose unfalsifizierbar = Marketing-Ballast** **- Bei Falschheit: Reputations-Schaden für Marcel**	Hook: "FluxAgent claims 300% — die Frage ist, gegen welche Workload." Body: "Was wir wissen: Repository angeblich MIT-licensed, Rust-Stack. Was die Quelle behauptet: 50.000 Devs, YC-W25. Beides nicht unabhängig verifiziert. Sub-Second-Orchestration ohne Definition — sub-second was? Bevor du migrierst: prüfe das Repo, die Benchmarks selbst." CTA: "Eigenes Test, dann Urteil. Nicht andersrum." Eigenschaften: - Single-Source-Claims sind sichtbar als solche - Hedging-Wörter ("claims", "angeblich") schützen Marcel - Audience wird zum eigenen Testen aufgefordert - Bei Falschheit der Quelle: Marcel hat sich nicht festgelegt - Bei Wahrheit: trotzdem informativer Reel

6. Lessons aus dem Stress-Test

Was der Test zeigt:

- V5.0 produziert SAUBEREN OUTPUT — der trotzdem 70% potentielle Falsch-Information enthält. Die Pipeline ist intern konsistent aber extern nicht validiert. Genau der GIGO-Effekt.
- V5.1's Agent 0 fängt die Red-Flags BEVOR sie zu Behauptungen werden. Die nachgelagerten Agenten bauen mit Wissen über die Trust-Lage. Output ist defensiver, aber auch verteidigungsfähig.

- Cross-Check-Direktiven sind manuelle Arbeit. Agent 0 listet sie auf, ABER ein Mensch (oder in V6: ein Multi-Modal-Adapter) muss sie tatsächlich ausführen. Ohne diese Schließung der Schleife ist V5.1 immer noch luckless.
- Der schleim_eval als Tertiary-Check fängt nur die offensichtlichen Sprach-Marker. Er ist kein Auditor. Wahrheits-Validierung passiert in Agent 0, nicht im Token-Filter.
- V5.0 hätte mit dem selben Source einen viralen Reel produzieren können — der bei Faktencheck als Falschnachricht entlarvt würde. Das ist Reputation-Risk, kein Pipeline-Bug.

Was der Test NICHT zeigt:

- Wie sich die Pipeline verhält bei einer Quelle MIT VERIFIZIERBAR FALSCHEN Inhalten (z.B. 'Bitcoin halbiert sich monatlich'). Hier müsste Agent 0's Faktencheck härter greifen.
- Performance-Charakteristik bei sehr großen Quellen (>10000 Worte). Crawler-Tokenkosten könnten unverhältnismäßig werden.
- Wie Auditor-Federation bei wirklich kontroversem Material (politische Aussagen, ethische Trade-offs) abstimmt. Hier ist 2/3-SOFT_CLEARANCE wahrscheinlicher als 3/3.

7. Konklusion

Der Stress-Test bestätigt die zentrale These der V5.1-Spec: OHNE Source-Validation ist eine sonst saubere Pipeline garbage-amplifying. V5.0 hat das Output-Audit-Problem gelöst, das Input-Audit-Problem geerbt.

V5.1 schließt die Lücke konzeptuell. V6 würde sie operativ schließen, weil dort die Cross-Check-Direktiven nicht nur aufgelistet, sondern via Multi-Modal-Adapter (GitHub-API, Web-Search, Citation-Lookup) automatisch ausgeführt werden.

Marcels Risiko in V5.0-Mode: ein Reel der intern sauber aussieht aber extern erstklassiges Fact-Check-Material darstellt. In V5.1-Mode: ein Reel der bewusst hedged und damit den Verteiler schützt, egal ob die Quelle hält.

V6 — Orchestrated Logic Firewall

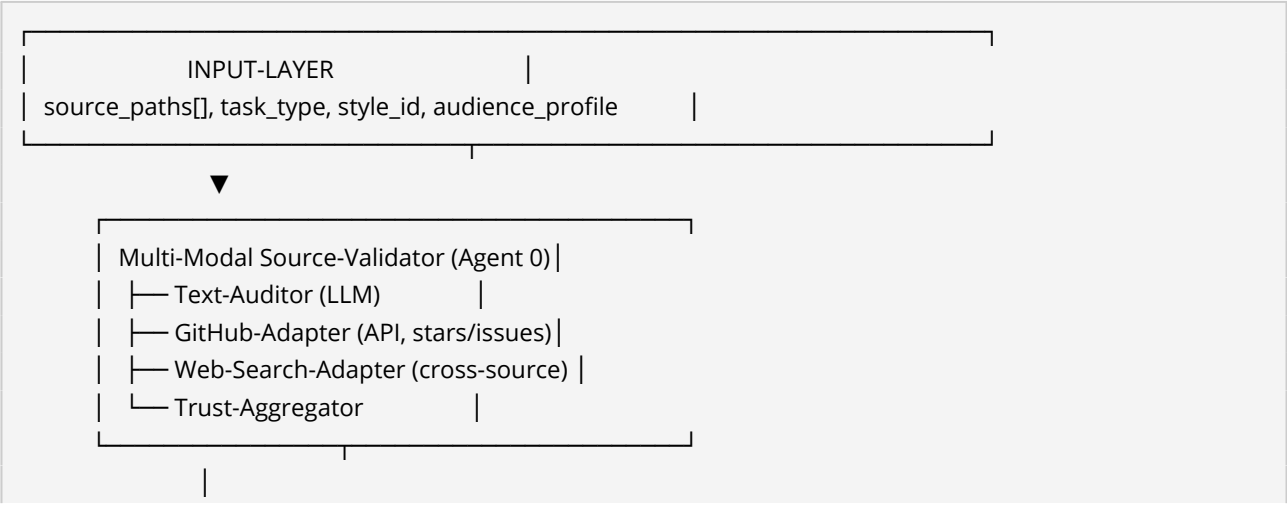
Vom Workflow-Dokument zur prüfbaren Pipeline — Architektur-Sketch

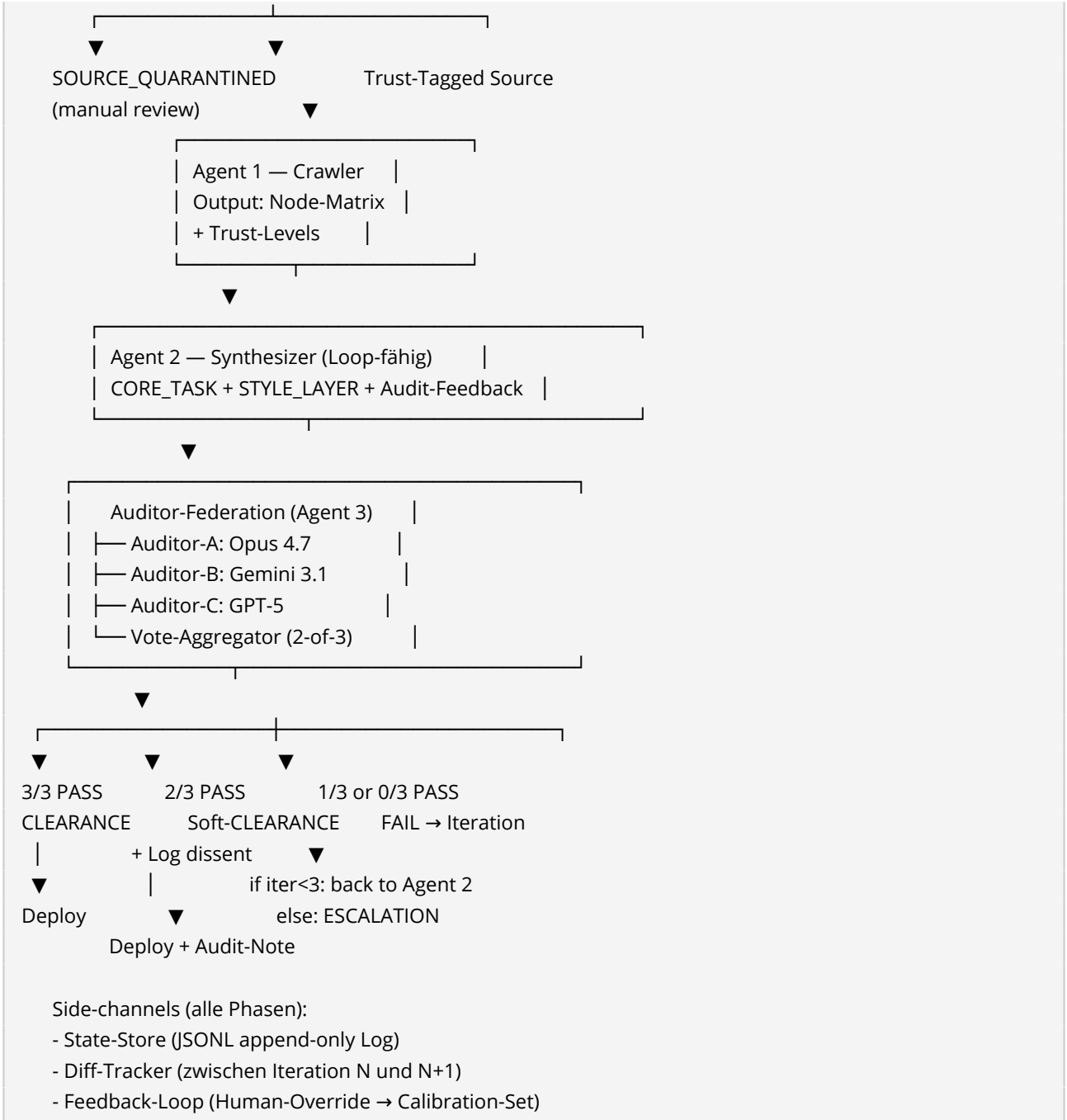
V5.1 ist konzeptuell vollständig, aber operativ noch eine Sammlung von Browser-Fenstern und Copy-Paste-Sequenzen. V6 ist der Schritt zu einer ausführbaren Pipeline mit Persistenz, Federation und automatischer Cross-Validation. Diese Ausarbeitung ist Architektur-Sketch + Implementations-Skeleton, kein Vollausbau.

1. Was V5.1 noch fehlt

V5.1 Limitation	V6 Antwort
Mensch orchestriert manuell — 3 Browser-Fenster, Copy-Paste, Iterations-Tracking im Kopf	Code-Orchestrierung — eine Pipeline-Run = ein Befehl. State + Audit-Trail persistent.
Ein Auditor entscheidet allein. Bias des Auditor-Modells kontaminiert Pipeline.	Auditor-Federation — Panel aus 3 verschiedenen Modellen, Vote-basiert. Split-Decisions → Human.
Source-Audit (Agent 0) ist textuell. GitHub-Daten, Update-Datum, Benchmark-Zahlen werden nicht maschinell geprüft.	Multi-Modal Source Validation — Adapter für GitHub-API, Web-Search, Citation-Lookup.
Zwischen Iterationen verschwinden Informationen. Agent 2 löscht manchmal Gutes statt nur Schlechtes zu fixen.	Diff-Tracking pro Iteration. Human sieht WAS Agent 2 geändert hat, nicht nur das Endresultat.
Calibration-Set ist statisch. System lernt nicht aus Human-Override-Entscheidungen.	Feedback-Loop — Human-Overrides werden zu neuen Calibration-Examples (versioned).
Templates für Reel/Tutorial/Comparison müssen jedesmal aus Prompts neu zusammengebaut werden.	Template-Library + versioned Style-Layers — YAML/JSON Configs statt Prompt-Strings.

2. Architektur-Übersicht





3. Orchestrator Code-Skeleton (Python, ~120 LOC)

Konkrete Implementierung — nicht Production-Ready, aber lauffähig nach Anpassung. Basis: requests + anthropic + openai SDKs. Keine externe LangGraph-Dependency nötig.

```
from dataclasses import dataclass, field
from typing import Optional, Dict, List
from pathlib import Path
```

```
import json, time, hashlib
import anthropic, openai  # google-generativeai analog

@dataclass
class PipelineRun:
    run_id: str
    source_paths: List[str]
    task_type: str          # "reel", "tutorial", "comparison"
    style_id: str           # "elite_focuscurated_v1.2"
    audience: str
    state_log: List[Dict] = field(default_factory=list)

    def log(self, stage: str, payload: Dict):
        self.state_log.append({
            "ts": time.time(),
            "stage": stage,
            **payload,
        })
        Path(f"runs/{self.run_id}.jsonl").open("a").write(
            json.dumps(self.state_log[-1]) + "\n"
        )

def agent_0_source_audit(run: PipelineRun) -> Dict:
    """Multi-Modal: combine LLM-Text-Audit + external adapters."""
    text_audit = _llm_call("opus-4-7", _load_prompt("agent_0_text"), run.source_paths)
    gh_signals = {}
    for claim in text_audit["red_flags"]:
        if claim.get("type") == "github_tool":
            gh_signals[claim["id"]] = _github_adapter(claim["repo"])
    web_signals = _web_search_adapter(text_audit["cross_check_directives"])
    aggregated = _trust_aggregator(text_audit, gh_signals, web_signals)
    run.log("agent_0", aggregated)
    if aggregated["quarantined"]:
        raise SourceQuarantined(aggregated["reason"])
    return aggregated

def agent_1_crawl(run: PipelineRun, trust_data: Dict) -> Dict:
    matrix = _llm_call("gemini-3-1", _load_prompt("agent_1_crawler"),
                       run.source_paths, context=trust_data)
    run.log("agent_1", {"node_count": len(matrix["nodes"])})
    return matrix

def agent_2_synthesize(run: PipelineRun, matrix: Dict,
                       audit_feedback: Optional[Dict] = None) -> str:
    core_task = _load_template(run.task_type)
    style = _load_style(run.style_id)
    draft = _llm_call("sonnet-4-6", _build_prompt(core_task, style),
                       context={"matrix": matrix, "feedback": audit_feedback})
```

```
    run.log("agent_2", {"iter": _iter_count(run), "len": len(draft)})
    return draft

def agent_3_federation(run: PipelineRun, draft: str, matrix: Dict) -> Dict:
    """Three independent auditors. Vote-based."""
    results = {}
    for auditor_id, model in [("A","opus-4-7"),("B","gemini-3-1"),("C","gpt-5")]:
        results[auditor_id] = _llm_call(
            model, _load_prompt("agent_3_firewall"),
            context={"draft": draft, "matrix": matrix}
        )
    pass_count = sum(1 for r in results.values() if r["clearance"])
    verdict = {
        "pass_count": pass_count,
        "results": results,
        "status": "CLEARANCE" if pass_count == 3 else
                  "SOFT_CLEARANCE" if pass_count == 2 else
                  "FAIL",
    }
    run.log("agent_3", verdict)
    return verdict

def run_pipeline(run: PipelineRun, max_iter: int = 3) -> str:
    """Top-level orchestrator with retry-cap and escalation."""
    _calibrate_auditors()  # runs Calibration-Set first
    trust = agent_0_source_audit(run)
    matrix = agent_1_crawl(run, trust)
    feedback = None
    last_draft = None
    for i in range(max_iter):
        draft = agent_2_synthesize(run, matrix, feedback)
        if last_draft is not None:
            run.log("diff", _diff(last_draft, draft))
        verdict = agent_3_federation(run, draft, matrix)
        if verdict["status"] in ("CLEARANCE", "SOFT_CLEARANCE"):
            return draft
        feedback = verdict["results"]
        last_draft = draft
    _escalate(run)
    raise PipelineEscalation(run.run_id)
```

4. Auditor-Federation — Vote-basierte CLEARANCE

V5.1 nutzt 1 Auditor-Modell. V6 nutzt 3 verschiedene Modelle als Panel. Vote-Outcomes:

```
3/3 PASS  →  CLEARANCE            (deploy frei)
2/3 PASS  →  SOFT_CLEARANCE       (deploy mit Audit-Note, Dissent geloggt)
1/3 PASS  →  FAIL (back to Agent 2, Feedback = Mehrheits-Audit)
0/3 PASS  →  FAIL (klar)

Warum 3 verschiedene Modelle:
  - Opus 4.7   stark in Logic-Konsistenz, Long-Context
  - Gemini 3.1 stark in Faktenprüfung gegen Tool-Use
  - GPT-5      stark in Sprach-Sensorik / Floskel-Detection

Bias-Korrelation zwischen verschiedenen Trainings-Frames ist niedriger als
innerhalb desselben Models. Was Opus übersieht, fängt Gemini auf und umgekehrt.

Dissent-Beispiel:
  Auditor A (Opus) PASS
  Auditor B (Gemini) PASS
  Auditor C (GPT) FAIL — Begründung: "Adverb-Overuse in §2"
  → SOFT_CLEARANCE, Audit-Note an Output gehängt für menschliche Sichtung
```

5. State-Persistenz + Diff-Tracking

Jeder Pipeline-Run schreibt JSONL-Append-Log nach `runs/{run_id}.jsonl`. Jeder Stage-Event (agent_0 / agent_1 / agent_2 / agent_3 / diff / escalation) bekommt einen Eintrag mit Timestamp und Payload.

```
Beispiel runs/2026-05-14-r042.jsonl:

{"ts": 1778763002, "stage": "agent_0", "verified": 12, "red_flags": 3, "quarantined": false}
{"ts": 1778763050, "stage": "agent_1", "node_count": 47}
{"ts": 1778763140, "stage": "agent_2", "iter": 1, "len": 2104}
{"ts": 1778763200, "stage": "agent_3", "pass_count": 1, "status": "FAIL"}
{"ts": 1778763280, "stage": "agent_2", "iter": 2, "len": 1987}
{"ts": 1778763289, "stage": "diff", "added_chars": 230, "removed_chars": 347, "moved_lines": 4}
{"ts": 1778763360, "stage": "agent_3", "pass_count": 3, "status": "CLEARANCE"}

Verwendung:
  - Debugging: warum hat Run X 3 Iterationen gebraucht?
  - Drift-Detection: pass_count Verteilung über letzte 100 Runs
  - Auditor-Calibration: welcher Auditor stimmte am häufigsten ab?
```

6. Multi-Modal Source-Validation (Adapter)

Agent 0 in V5.1 ist rein LLM-basiert. V6 ergänzt externe Adapter, weil reine LLM-Text-Audits blind sind für externe Faktoren (Repo-Aktivität, Citation-Count, Source-Update-Datum).

```
class TrustAdapter:
    def check(self, claim: dict) -> dict:
        """Returns trust_delta (-1.0 to +1.0) + reason."""
        raise NotImplementedError

class GitHubAdapter(TrustAdapter):
    """Wenn Claim ein GitHub-Tool referenziert: prüfe Stars, Issues, Last-Commit."""
    def check(self, claim):
        repo = claim.get("repo")
        meta = github_api(f"/repos/{repo}")
        signals = {
            "stars": meta["stargazers_count"],
            "issues_open": meta["open_issues_count"],
            "last_commit_days_ago": _days_since(meta["pushed_at"]),
        }
        # Heuristik
        delta = 0.0
        if signals["last_commit_days_ago"] > 180: delta -= 0.3
        if signals["stars"] < 100: delta -= 0.2
        if signals["issues_open"] > signals["stars"] * 0.5: delta -= 0.4
        # Search issues for negative reports
        neg_issues = github_search(f"repo:{repo} doesnt work OR broken OR wrong")
        if len(neg_issues) > 5: delta -= 0.5
        return {"trust_delta": delta, "signals": signals,
                "negative_issues": neg_issues[:5]}

class WebSearchAdapter(TrustAdapter):
    """Cross-source check: behauptet die Quelle was niemand sonst behauptet?"""
    def check(self, claim):
        results = search_web(f"\"{claim[\"text\"]}\"")
        unique_sources = len(set(r["domain"] for r in results))
        delta = 0.0
        if unique_sources < 2: delta -= 0.6   # Single-source claim
        if unique_sources > 10: delta += 0.2  # widely cited
        return {"trust_delta": delta, "source_count": unique_sources}

class CitationAdapter(TrustAdapter):
    """Wenn Claim ein Paper zitiert: prüfe Citation-Count, Retraction-DB."""
    def check(self, claim):
        paper = claim.get("doi")
        if not paper: return {"trust_delta": 0.0}
        meta = openalex_lookup(paper)
        delta = 0.0
        if meta.get("is_retracted"): delta -= 1.0  # hard penalty
        if meta["cited_by_count"] > 100: delta += 0.3
        return {"trust_delta": delta, "citation_count": meta["cited_by_count"]}
```

7. Template-Library + Versioned Style-Layers

V5.1 hat STYLE_LAYER als Prompt-String. V6 macht daraus YAML-Configs mit Versionierung.

```
templates/reel.yaml:
  core_task: |
    Erstelle ein 30-Sekunden Reel-Skript aus der Node-Matrix.
    Struktur: Hook (2s) → Body (24s, max 3 Punkte) → CTA (4s).
    Jeder Punkt referenziert mindestens einen VERIFIED-Node.
  required_fields: [hook, body_points, cta]
  max_length_chars: 600

styles/elite_focuscurated_v1.2.yaml:
  name: "Elite Focus Curated"
  version: "1.2"
  changelog:
    - v1.2 (2026-05-14): "Adverb-Cap auf max 2 pro 100 Worte"
    - v1.1 (2026-04-30): "Hook muss mit Zahl oder Frage starten"
    - v1.0 (2026-04-15): "Initial release"
  rules:
    - "Direkter, technischer Tonfall. Imperative bevorzugt."
    - "Keine Adjektiv-Inflation (revolutionär/disruptiv/etc.)."
    - "Zielgruppe: Solo-Founder, AI-Builder. Annahme: Fachbegriffe bekannt."
  forbidden_words: [revolutionär, disruptiv, game-changer, deep-dive,
                    paradigmenwechsel, holistisch, synergie]
  required_patterns:
    - "Mindestens 1 Code-Block oder Zahl pro 200 Worte"

Pipeline-Aufruf:
  lfw run --source paper.pdf,issue_thread.md \
          --task reel \
          --style elite_focuscurated_v1.2 \
          --audience "solo_founders_ai"
```

8. Feedback-Loop in Calibration-Set

Wenn Mensch nach Escalation eine Entscheidung trifft (Override, Akzept, Re-Brief), wird diese Entscheidung als neuer Calibration-Example gespeichert. System lernt über Zeit, was Mensch als gut/schlecht eingestuft hat — ohne Re-Training.

```
calibration_set/2026-05-14_r042.yaml:
  source_text: "..."  # 200 char excerpt
  draft_iter3: "..."  # the failed iteration text
```

```
  auditor_verdicts:
    A: {clearance: false, fail_at: "[1] FLOSKEL-CHECK"}
    B: {clearance: false, fail_at: "[1] FLOSKEL-CHECK"}
    C: {clearance: true}
  human_decision: "OVERRIDE_TO_PASS"
  human_reason: |
    "Wort \"breakthrough\" wurde von A+B geflagged, aber im Kontext
    direkt zitiert aus Original-Quelle. Hedging-Anführungszeichen vorhanden."
  lesson: |
    "FLOSKEL-CHECK muss erkennen ob ein verbotenes Wort als wörtliches
    Zitat (mit Anführungszeichen) verwendet wird → dann erlauben."

Bei nächstem Auditor-Calibration-Run wird dieser Fall mitgeführt.
Wenn Auditor erneut den Fall falsch klassifiziert → Prompt schärfen.

Effekt:
  - Calibration-Set wächst organisch mit echten Edge-Cases
  - System wird über Zeit besser, ohne dass das User selbst Prompt-Engineering macht
  - Bei Auditor-Modell-Update (Opus 4.7 → 4.8): Calibration zeigt sofort wo neuer
    Modell-Stand vom alten abweicht
```

9. Migration V5.1 → V6

V6 ist kein Re-Build. Es ist V5.1 mit Code drumherum. Die Prompts aus V5.1 werden weiterverwendet und in `prompts/` als versionierte Files abgelegt.

```
Schritt 1 (Tag 1):
  - V5.1-Prompts in prompts/*.md ablegen (versionierbar)
  - Simple Python-Runner schreiben der die 4 Agenten sequenziell aufruft (~50 LOC)
  - State-Log als JSONL aktivieren
  → Du hast: V5.1 + Orchestrierung, noch kein Federation

Schritt 2 (Tag 2-3):
  - Auditor-Federation aufsetzen (3 LLM-Provider-Keys, 1 Vote-Aggregator)
  - Calibration-Set initialisieren (4 Examples reicht)
  → Du hast: V6 Core ohne Multi-Modal

Schritt 3 (Tag 4-5):
  - GitHub-Adapter implementieren (PyGithub-Lib, ~30 LOC)
  - Web-Search-Adapter (Brave/Tavily/Serper API)
  - Trust-Aggregator vereinigen
  → Du hast: V6 mit Multi-Modal Agent 0

Schritt 4 (Tag 6-7):
  - Template-Library + Style-YAMLs (Reel, Tutorial, Comparison)
```

```
- CLI-Tool als Entry-Point (typer oder click)
- Feedback-Loop Integration
→ Du hast: V6 Komplett
```

10. Eigenschaften der V6-Pipeline

Was V6 von V5.1 unterscheidet — gemessen in System-Eigenschaften, nicht Features:

```
Reproduzierbarkeit:    JEDER Run loggt JSONL, mit run_id reproduzierbar
Auditierbarkeit:       Mensch kann Run-History inspizieren, Fehler-Quellen orten
Cost-Awareness:        Multi-Modal Adapter nur bei Red-Flag-Claims (Budget-Optimierung)
Drift-Detection:       pass_count-Statistik über 100 Runs zeigt Modell-Verschiebungen
Self-Improvement:      Calibration-Set wächst mit Human-Overrides
Multi-Tenant-fähig:    Style-Versions getrennt von Core-Logic → mehrere Brands
Cross-Modell:          Keine Lock-In auf einen Provider (Anthropic/Google/OpenAI parallel)
```

V6 ist ein System. V5.1 ist ein Workflow-Doc. V5.0 ist ein Prompt. V4 war ein Tweet. Linear progression in operativer Härte.