ChatGPT

Technisch Ontwerp Mustika Rasa-systeem

Inleiding

Het Mustika Rasa-systeem is een lokaal draaiende architectuur voor de vertaling en annotatie van het historische Indonesische kookboek *Mustika Rasa*. Doel is een **betrouwbare digitale editie** te creëren die niet alleen een "mooie vertaling" oplevert, maar vooral een **traceerbaar proces** waarbij elke wijziging verantwoord en reproduceerbaar is [1] [2]. Het ontwerp volgt de principes *"Redactie leidt, techniek ondersteunt, governance beschermt"* [3]. Dit betekent dat menselijke **redactionele beslissingen leidend** zijn, de technische componenten (AI-agents, tools, pipelines) deze beslissingen **ondersteunen maar niet zelf nemen**, en een gelaagd **governance-raamwerk** bewaakt dat alle veranderingen controleerbaar en verantwoord verlopen.

We baseren ons op de bestaande TOGAF-geïnspireerde architectuurschets van het project [4]. In lijn hiermee beschrijft dit document de architectuur in lagen: de **Business/Governance-laag** (redactionele doelen, governance structuren), de **Applicatie-laag** (agents, orkestratie, workflows), de **Data-laag** (opslag van teksten, annotaties en meta-informatie) en de **Technologie-laag** (infrastructuur en tools op de Mac Mini). Centraal staan de projectdoelen van **integriteit, transparantie en herleidbaarheid** – het systeem moet altijd kunnen aantonen: *"zo was het — zo is het — en dit is waarom het veranderde."* [5] [6].

Deze architectuur is ontworpen voor een Mac Mini met minimale overhead en kosten. We maken zoveel mogelijk gebruik van lokale tooling en open-source AI-modellen (zoals **Mistral** via **Ollama**) om afhankelijkheid van dure externe APIs te vermijden. De **Codex CLI** en **Crew**AI libraries worden ingezet voor orkestratie en promptuitvoering in de terminal-omgeving, zodat het geheel lokaal, scriptbaar en schaalbaar blijft. *Menselijke inbreng blijft essentieel:* automatisering wordt maximaal benut voor analyses en voorstellen, maar de mens blijft in de lus voor alle betekenisvolle beslissingen en kwaliteitstoetsing.

Governance en Kernprincipes

Het systeem opereert strikt binnen het gelaagde governance-model van het project (**GOV-A / GOV-B / BRIDGE**). In deze structuur is **GOV-A** de project governance: alle ontwerpbeslissingen, strategieën en workflows worden door mensen bepaald buiten runtime om [7]. **GOV-B** is de systeem-governance tijdens runtime: formele regels en validators in de software die afdwingen wat agents mogen produceren en wat als canoniek wordt geaccepteerd [8]. De **BRIDGE-laag** verbindt de intenties van GOV-A met de handhaving in GOV-B: dit betreft beleidsdocumenten en ontwerpen (zoals dit document, *autonomy envelopes*, human gate policies) die de kaders schetsen, **zonder zelf uitvoerbare code te zijn** [9]. We introduceren **geen nieuwe autoriteits- of canon-mechanismen** buiten deze structuur; alle systeemregels zijn verankerd in bestaande governance-artikelen of expliciet goedgekeurde uitbreidingen. Documenten in de repository krijgen ook een label welke laag ze betreffen (GOV-A, GOV-B of BRIDGE) conform het classificatieschema [10] [11].

Een paar centrale principes uit de governance-documenten sturen het ontwerp:

- **Agents zijn signaleerders, niet beslissers:** Agents voeren analyses uit, signaleren afwijkingen/ambiguiteten en doen voorstellen, maar nemen **nooit autonome, onomkeerbare inhoudelijke beslissingen** [12]. Ze *signaleren, analyseren, annoteren en adviseren*, maar alle betekenisvolle wijzigingen vereisen een menselijke beslissing via een **Human Gate** [13]. Dit waarborgt dat de uiteindelijke autoriteit bij menselijke redacteuren blijft en voorkomt dat AI ongemerkt de inhoud verandert.

- **Automatisering met menselijke toetsing:** Het systeem maximaliseert automatisering (AI ondersteunt bij repetitieve detectie- en analyse-taken), maar bij **belangrijke onzekerheden, culturele nuances of risico's wordt altijd geëscaleerd naar een mens**. Agents volgen het motto: *"documenteer → signaleer → escaleer; niet: interpreteren → beslissen → corrigeren"* [14]. Bij twijfelgevallen geldt het rigide **voorzorgsprincipe**: *documenteren en defereren, niet autonoom doorvoeren*. In de agent-prompts is expliciet opgenomen: *"bij twijfel: documenteer, beslis niet … nooit 'best guess'"* [15].

- **Geen stille wijzigingen:** Elke geconstateerde afwijking, correctie of ingreep moet **zichtbaar en herleidbaar** zijn. Agents zullen nooit op eigen houtje tekst "verbeteren" zonder dit als voorstel vast te leggen. Governance-richtlijnen stellen dat bij twijfelgevallen of mogelijke impact *"geen stilzwijgende keuzes"* gemaakt mogen worden [16] – in plaats daarvan wordt een ambigue kwestie gemarkeerd en gedocumenteerd met bijvoorbeeld een `[AMBIGUOUS]` tag en een toelichting, zodat duidelijk is dat dit punt openstaat voor besluitvorming.

- **Traceerbaarheid en auditability:** Het systeem onderhoudt een volledig **documentair spoor** van oorspronkelijke bron, wijzigingen en rationale. Dit wordt bereikt via uitgebreid logging, versiebeheer en unieke identificatie van artefacten. Het principe *"oorsprong → huidige staat → rationale"* is in alle lagen doorgevoerd [17]. Beslissingen worden vastgelegd met verwijzing naar de onderliggende governance-bronnen en de logs [18]. Hierdoor moet later elke beslissing herleidbaar zijn: **wat is gewijzigd, door wie/wat, op basis van welke reden of bewijs**.

- **Soft-stops en Human Gate:** Het ontwerp bevat expliciete punten waar de workflow **pauzeert voor menselijke input** wanneer vooraf gedefinieerde omstandigheden optreden. Bijvoorbeeld: als een agent een potentieel **cultureel gevoelig begrip of gezondheidsadvies** tegenkomt, wordt dit gemarkeerd als een **Gate**-moment dat niet automatisch gepasseerd mag worden [19] [20]. Via een *Human Gate* moet een editor expliciet beoordelen hoe hiermee om te gaan (annoteren, toelichten of alsnog doorlaten). Zolang een gate niet door een mens is geadresseerd, blijft de status *"in afwachting"* en gaat de automatisering niet verder met dat onderdeel.

- **Governance-STOP bij non-compliance:** Het systeem handhaaft bepaalde niet-onderhandelbare voorwaarden. Bijvoorbeeld, **verliest een voorstel zijn herkomst/provenance of ontbreekt verplichte documentatie, dan grijpt governance in met een STOP** [21] [22]. Een "governance-stop" betekent dat de geautomatiseerde workflow *direct stopt* voor het betreffende onderdeel tot het probleem is opgelost. Dit voorkomt dat er resultaten ontstaan die niet voldoen aan de governance-eisen (zoals ongedocumenteerde wijzigingen of ongetraceerde inhoud).

Deze principes vormen samen de **veiligheidsrails** van het systeem: ze zorgen dat het gebruik van AI de redactionele integriteit niet ondermijnt. In de volgende secties vertalen we deze principes naar concrete architectuurcomponenten, data-indeling, workflows en mechanismen.

Architectuur Overzicht

In grote lijnen bestaat het Mustika Rasa-systeem uit een reeks **gespecialiseerde AI-agents** die gecoördineerd worden door een centrale **Orchestrator**, in nauwe samenspraak met menselijke redacteuren. Ieder stuk content (bijvoorbeeld een recept of paragraaf) doorloopt een gestandaardiseerd proces alsof het een **redactionele pijplijn** is. Deze pijplijn is vergelijkbaar met een menselijke editorial workflow, maar dan ondersteund door AI:

- **Orchestrator:** Een centrale coördinerende module (een soort meta-agent) die bepaalt *welke agents in welke volgorde* op een bepaald contentobject worden losgelaten. De Orchestrator waakt over het proces, start de agents op, verzamelt hun output en bewaakt dat alle governance-regels (zoals stops en escalaties) worden gerespecteerd [23] . De Orchestrator is ook verantwoordelijk voor het opvolgen van templates/protocol (zodat elke agent met de juiste prompt en parameters draait) en het afkappen van het proces als een harde STOP-conditie wordt geraakt. In implementatietermen kan de Orchestrator een **CrewAI "flow"** zijn die de taken orkestreert volgens vooraf gedefinieerde stappen.

- **Specialistische Agents:** Voor verschillende **redactionele deelgebieden** zijn er aparte agents (zie ook *capabilities* verderop). Enkele voorbeeldrollen:

- *Vertaling-agent:* controleert of de vertaling consistent is en signaleert mogelijke betekenisverschillen tussen bron (Indonesisch) en vertaling (Engels).
- *Annotatie-agent:* doet voorstellen voor voetnoten of uitleg bij culturele termen, ambiguïteiten of historische context.
- *Terminologie-agent (Glossary):* bewaakt terminologische consistentie (bv. detecteert dat "lobak" soms als "radish" en soms als "daikon" is vertaald) en legt voorstel-items aan in een glossary-lifecycle.
- *Veiligheidsagent:* let op gezondheidsclaims of potentieel gevaarlijke kookinstructies en markeert die voor nader review (bijv. `[SAFETY_WARNING]`).
- *Continuïteit/Coherentie-agent:* spoort inconsistenties in stijl of structuur tussen recepten op (bv. volgorde van stappen, of een recept dat een ingrediënt noemt dat nooit gebruikt wordt).
- *OCR/Structuur-agent:* (voor vroege fase) checkt de integriteit van de gedigitaliseerde bron (tabellen correct overgenomen, paginanummering, ontbrekende tekst).
- … (Deze lijst kan groeien; de Vision-doc noemt bijv. agents voor *Fidelity, Readability, Design, Image* etc. [24]).

Elke agent is **beperkt tot één duidelijk afgebakende taak** en werkt volgens een strikt prompt- en outputcontract (zie hieronder bij *Promptstructuur*). Ze lezen de relevante input (bv. een recepttekst, eventueel met context) en genereren **uitsluitend een gestructureerde rapportage** van hun bevindingen: meestal in JSON of markdown-formaat met labels, *geen* vrije tekst wijzigingen. Cruciaal: agents *mogen de bron niet rechtstreeks aanpassen*. In plaats daarvan leggen zij hun *bevindingen als voorstel* neer in een **niet-**

canonieke laag (de sandbox), zoals “Recept X: term Y is ambigu, mogelijke interpretaties …”. Hiermee functioneren zij als de “voelsprieten” van het systeem, niet als de handen die de tekst aanpassen [25].

- **Menselijke Redacteur (Human-in-the-Loop):** Na afloop van een agent-run (of tussentijds bij een gate) komt de menselijke expert in beeld. Deze persoon (of redactie-team) bekijkt de **gegenereerde voorstellen, annotaties en waarschuwingen** van de agents. De redactie hakt knopen door: welke suggesties worden overgenomen, welke worden verworpen, welke kwesties blijven voorlopig openstaan? Bij elk zo’n beslissing zorgt de redactie voor de juiste **closure**-registratie: bv. een voorstel accepteren met een korte rationale in een noot, of een issue als “blijft ambigu” markeren. De mens is dus de **gatekeeper** die alle signalen verwerkt tot definitieve acties. Geen enkel voorstel wordt canoniek (onderdeel van de gepubliceerde tekst) zonder menselijke goedkeuring. Dit is in lijn met het *Human Gate*-principe: *“Meaning changes, glossary resolution, cultural interpretation,… → Human Gate mandatory”* [26] [27].

- **Validators & Governance Agents:** Naast de inhoudelijke agents zijn er enkele ondersteunende mechanismen die continu de **compliance** in de gaten houden. Denk aan een *Governance-monitor* (dit kan een script of eenvoudige agent zijn) die de output van agents naloopt op bijvoorbeeld het voorkomen van verboden acties of velden. Bijvoorbeeld, als een agent toch een vertaalde zin als “verbetering” probeert uit te geven (wat tegen de regels is [28]), zal de validator dit detecteren en een *STOP* triggeren voordat het verder kan. Evenzo kan een *Methodology & Audit agent* toezien dat alle vereiste logvelden aanwezig zijn en dat er een rationale is gegeven waar nodig [23]. Deze governance-hulpmiddelen vormen samen de **GOV-B** laag tijdens runtime: ze zijn geprogrammeerd om *PASS/FAIL* situaties te signaleren en de Orchestrator te laten ingrijpen indien nodig [29]. Een FAIL van zo’n validator leidt tot een onmiddellijke governance-stop en eventueel escalatie naar de mens om het op te lossen, conform het *stop-model*.

Gezamenlijk vormen deze componenten een **hybride mens-AI workflow**. Een typische verwerking van één recept gaat bijvoorbeeld als volgt: De Orchestrator start de verschillende agents één voor één. Eerst meldt de *OCR/structuur-agent* eventuele technische fouten; daarna identificeert de *Vertaling-agent* betekenisverschillen; de *Terminologie-agent* highlight inconsistent gebruikte termen; de *Veiligheidsagent* roept een gate aan voor een twijfelachtig medisch advies in de tekst, waarop de Orchestrator de run pauzeert. De redactie bekijkt dit signaal, beslist bijvoorbeeld om een voetnoot toe te voegen dat dit advies alleen historisch is, logt die beslissing en laat de Orchestrator doorgaan. Na afloop ligt er een pakket van annotatievoorstellen en logbestanden klaar. De redactie verwerkt alle voorstellen in de canonieke tekst (in de vorm van annotaties, aanpassingen of besluiten om niets te doen) en sluit de kwestie af met documentatie. Dit proces waarborgt dat **elk signaal een closure krijgt** [30] [31]: geaccepteerd (met toelichting), expliciet uitgesteld, verworpen (met uitleg) of bewust open gelaten, maar niets verdwijnt stilzwijgend.

(Diagrambeschrijving: In de TOGAF-architectuurschets komt dit overeen met een Application Architecture waarin meerdere “agent services” onder regie van een central orchestration component samenwerken, verbonden met een Business layer van menselijke governance, en onderbouwd door een Data layer voor content & logs. Het systeem is modulair: nieuwe agents (capabilities) kunnen worden toegevoegd zonder het kernprincipe te schenden, omdat de orchestrator en governance-laag hun gedrag kaderen.)

Data- en Opslagarchitectuur

Een helder onderscheid tussen **canonieke** en **niet-canonieke** data is fundamenteel voor dit systeem. Canonieke data is alles wat de uiteindelijke gepubliceerde inhoud of officiële besluitvorming weerspiegelt. Niet-canonieke data is tijdelijk, experimenteel of louter ondersteunend (bv. agentvoorstellen, tussenresultaten). We scheiden deze strikt in de directorystructuur en in de manier waarop ermee omgegaan wordt.

Canonical vs Non-Canonical lagen: De volgende tabel (afgeleid van de governance-richtlijnen) toont de verschillende lagen en hoe agents hiermee mogen interacteren:

Laag	Rol in het systeem	Agent-editing toegestaan?
Facsimile scans / OCR output	Historisch bronmateriaal	*nee* (read-only)
Gecorrigeerde OCR-tekst	Canonieke werkttekst	*nee* (alleen lezen)
Editorial annotations (draft)	Voorstel-annotaties (niet-canoniek)	✔ *ja* (alleen sandbox)
Glossary (canoniek)	Referentielaag (begrippenkader)	*nee* (alleen via lifecycle)
Research notes	Onderzoeksnotities, voorstellen	✔ *ja* (sandbox)
Pilot artifacts	Documentaire tussenproducten	✔ *ja* (sandbox)

Bron: governance clarifications [32]. Regel: als een laag invloed heeft op wat lezers uiteindelijk zien, dan is het canoniek – tenzij expliciet als sandbox aangemerkt [33].

Concreet betekent dit dat de **hoofddocumenten** van het project (brontranscripties, definitieve vertalingen, uitgewerkte annotaties, glossary beslissingen, etc.) nooit door agents rechtstreeks worden aangepast. Agents werken uitsluitend in **sandbox-ruimtes**: aparte bestanden of tijdelijke kopieën waar ze hun voorstellen in plaatsen. Als een agent bijvoorbeeld een afwijking signaleert in een recept, zal het die vastleggen in een annotatievoorstel (bijv. een Markdown-bestand of JSON structuur in `sandbox/proposals/…`). Pas na menselijke review kan zoiets verplaatst worden naar de echte documentatie.

Directorystructuur: Onderstaand een overzicht van de voorgestelde directory- en bestandsstructuur waarin deze scheiding tot uiting komt:

```
project-root/
├── docs/  (canonieke documenten en resultaten)
│   ├── VISION_AND_STRATEGY.md
│   ├── WORKFLOW.md
│   ├── AGENTS.md
│   ├── CONTENT_ROADMAP.md
│   ├── 10-governance/        (governance policies, human gate regels etc.)
│   │    ├── GOV_A_B_CLASSIFICATION.md
│   │    ├── AGENT_AUTONOMY_ENVELOPE.md
```

```
│   │   └── ...
│   ├── source/               (bronmateriaal en vertaling)
│   │   ├── mustika_rasa_id.md   (originele tekst Indonesisch, canoniek)
│   │   ├── mustika_rasa_en.md   (lopende vertaling Engels, canoniek)
│   │   └── ...
│   ├── edition/              (wetenschappelijke editie output)
│   │   ├── annotated_chapter1.md   (bv. tekst + voetnoten)
│   │   └── ...
│   ├── glossary/             (canonieke glossary beslissingen)
│   │   └── glossary.md
│   ├── decisions/            (vastgelegde beslissingen, Phase-7+)
│   │   └── 2026/
│   │       └── lobak_case_decision001.md  (voorbeeld beslissing)
│   ├── CODEX_SESSION_LOG.md  (logboek van sessies/proposal moves)
│   └── ... (andere canonieke docs, e.g. styleguides)
├── sandbox/  (niet-canonieke uitvoer en tussenbestanden)
│   ├── agent_runs/       (volledige output van agent-runs per item)
│   │   └── RUN_20260104_003/
│   │       ├── annotator_primary.json   (output van Annotatie-agent)
│   │       ├── translator_primary.json  (output van Vertaling-agent)
│   │       └── ...
│   ├── run_logs/         (logfiles per run, menselijk leesbaar trace van
dialoog)
│   │   └── RUN_20260104_003.log
│   ├── proposals/        (losse voorstelbestanden, indien niet direct vanuit
JSON leesbaar)
│   │   └── lobak_term_proposal.md   (voorbeeld voorstel voor glossary opname)
│   └── research_notes/   (notities, hypotheses door agents of onderzoekers)
│       └── lobak_context_research.md
├── prompts/   (prompt templates voor agents)
│   ├── translator_agent.md
│   ├── annotator_agent.md
│   ├── glossary_agent.md
│   └── ...
├── logs/   (globale logbestanden voor governance)
│   ├── warnings.log        (algemene waarschuwingen en agent-issues)
│   ├── stop.log            (elke Governance-STOP gebeurtenis)
│   └── governance_notes.log (notities van governance of bijzonderheden)
├── src/    (orchestrator code, eventueel CrewAI flows, tools)
│   ├── orchestrator.py
│   ├── validators.py
│   └── ...
└── README.md
```

Toelichting: in `docs/` staan de formele projectdocumenten en resultaten. De submap `10-governance/` bevat de governance- en beleidsdocumenten (GOV-A en BRIDGE materialen). `source/` bevat de brontekst

en de lopende vertaling (canoniek omdat dit is wat uiteindelijk gepubliceerd wordt). `edition/` zou bijvoorbeeld de samengestelde wetenschappelijke editie kunnen bevatten (bijv. hoofdstukken met integrale annotaties). `glossary/` bewaart de beslissingen over termen. `decisions/` is gereserveerd voor formele beslissingrecords (zoals in Phase-7 voorzien in *CANONICAL_INDEX.md* [34] [35]).

De `sandbox/` directory is de werkruimte voor agents. Hier wordt per *run* van de Orchestrator een unieke run-map aangemaakt (`RUN_<YYYYMMDD>_<NNN>` formaat [36]) waarin alle outputs en logs voor dat specifieke item bewaard worden. Dit maakt het mogelijk de volledige context van een run te herbekijken of te debuggen. Binnen zo'n run-map kan output per agent in aparte JSON-bestanden staan (of in één gecombineerd JSON, afhankelijk van implementatie) – zo is de output **machine-parseable** en eenvoudig te valideren. Het `run_logs/` bestand is een leesbare transcript/log (bijvoorbeeld de gegenereerde tekstconversaties of debuginfo) voor menselijke inzage, indien nodig. De `proposals/` map kan gebruikt worden om specifieke voorstellen die uit meerdere JSON-velden bestaan, samen te vatten in een markdown-format voor gemakkelijker menselijke review (maar dit kan ook on-the-fly gepresenteerd worden in een UI of rapport). `research_notes/` is een optionele plek voor verdiepende informatie die door agents of mensen is verzameld buiten de formele annotaties om (en wordt als niet-canoniek beschouwd tenzij omgezet in besluitdocumenten).

Canonical moves: Een streng regime geldt voor het **overhevelen van inhoud van de sandbox naar canonieke documenten**. Volgens governance is dit *alleen toegestaan indien* (a) de provenance intact blijft, (b) de verhuizing gelogd wordt, en (c) de tekst geen impliciete besluitvorming veronderstelt [37] [38]. Vandaar het bestand `docs/CODEX_SESSION_LOG.md`: iedere keer dat een redacteur besluit een voorstel vanuit de sandbox te verwerken in de officiële documenten, wordt dit hier geregistreerd in een gestandaardiseerd formaat, bijvoorbeeld:

```
[PROPOSAL_MOVED]
from: sandbox/agent_runs/RUN_20260104_003/annotator_primary.json (proposal #2)
to: docs/edition/chapter2.md (voetnoot 12)
reason: accepted in editorial meeting 2026-01-05 (zie rationale in tekst)
```

Mocht ooit een voorstel per ongeluk zonder logging of bronverwijzing zijn overgenomen, dan geldt **Missing provenance → Governance-STOP**: de `stop.log` zou zo'n gebeurtenis noteren en het systeem zou verdere output blokkeren tot dit is opgelost [22]. Over het algemeen verwijderen we nooit zomaar sandbox-data; *"Deletion equals rollback, not cleanup"* [39] wil zeggen dat het verwijderen van sandbox runs alleen plaatsvindt als bewuste rollback (bijv. een hele run ongeldig verklaren), anders blijven ze bewaard voor audit-trail tot een expliciete archivering na review.

Indexering en trace: Om de traceerbaarheid te versterken, voorziet het ontwerp in index-bestanden die verbanden leggen tussen bronnen, voorstellen en beslissingen. Zo is in *CANONICAL_INDEX.md* beschreven dat vanaf Phase-7 elk formeel besluit een index-item krijgt met verwijzingen naar het relevante bronfragment, de reviewnotities en de AI-artefacten [34] [40]. In de huidige implementatiefasen zal dit wellicht handmatig of middels scripttooling gebeuren. Denk aan een eenvoudige **SQLite database of CSV** waarin ieder gesignaleerd issue een ID krijgt en velden voor status (open/deferred/closed), locatie (document + plaats in tekst), link naar run-output, en link naar closure (beslissing of annotatie). Dit vergemakkelijkt zowel *traceability* (je kunt alle voorstellen bijhouden en hun afhandeling) als *indexering* voor publicatie (bijv. een lijst van alle bijzondere termen en hoe ze zijn afgehandeld). Omdat we kosten en

complexiteit laag willen houden, kan in eerste instantie een markdown-overzicht volstaan waarin we per hoofdstuk of onderwerp de openstaande punten en beslissingen bijhouden. Op termijn kan dit uitgroeien tot een geautomatiseerde indexpagina die direct uit de logs en beslissingdocumenten wordt gegenereerd.

Prompt- en Outputstructuur van Agents

Om de output van agents consistent en controleerbaar te houden, hanteren we een **vast promptpatroon** (conform *Agent Prompt Pattern P5* [41]) voor alle AI-agents. Iedere agent-prompt bevat vaste secties, zodat duidelijk is wat de agent wel/niet moet doen en hoe het resultaat eruit moet zien. De hoofdsecties van elke prompt zijn:

1. **ROLE:** Één zin die de rol van de agent scherp afbakent – wat is de opdracht en scope.
2. **MISSION BOUNDARIES:** Duidelijke opsomming van *welke taken de agent uitvoert* en *welke niet*. Alles buiten dit mandaat moet het agent negeren of hooguit melden als "out-of-scope", **niet zelf oplossen** [42] .
3. **INPUT:** Specificatie van wat de agent ontvangt (bijv. "je krijgt de originele zin X en de vertaalde zin Y" of een heel recept) inclusief formaten, velden en randvoorwaarden. Dit voorkomt dat het model zelf aannames gaat doen over ontbrekende context.
4. **OUTPUT CONTRACT:** De precieze vereisten voor de output. Dit is **machine-checkbaar formaat** (idealiter JSON met vaste velden of een strak gedefinieerde Markdown structuur) [43] . Hier wordt bv. gedefinieerd dat de agent een lijst van gevonden verschillen moet teruggeven, elk met velden als `location`, `description`, `impact` (semantisch, cultureel, etc), en eventueel een veld `resolution_suggestion` dat vaak leeg blijft tenzij er een veilige suggestie is. Een klein voorbeeld of schema wordt in de prompt gegeven, zodat de agent weet hoe het eruit moet zien.
5. **RULES:** Een korte lijst (max ~6 regels) met harde instructies of tests die de agent altijd moet volgen [44] . Bijvoorbeeld: "Gebruik uitsluitend de velden uit het output-schema", "Geef geen verklaringen als je niet 100% zeker bent", "No translations or rewrites of the text" (als dat verboden is).
6. **UNCERTAINTY & ESCALATION:** Hier wordt herhaald dat bij enige twijfel de agent *niet zelf beslist*. De prompt benadrukt: *"bij twijfel: documenteer, beslis niet; gebruik desnoods* `NONE` *of markeer als* `AMBIGUOUS` *"* [15] . Ook wordt het mechanisme van soft-stop uitgelegd: de agent mag bij ernstige twijfel een speciaal signaal geven (bv. een veld `"escalation": "HUMAN_REVIEW_REQUIRED"`) zodat de Orchestrator weet dat hier een menselijke beoordeling moet volgen.
7. **FORBIDDEN:** Een expliciete lijst van wat de agent **niet mag doen** [28] . Voorbeelden: geen eigen vertalingen suggereren, geen "equivalent" termen invoeren, geen OCR-correcties doorvoeren, niet zelfstandig glossary-keuzes maken, en überhaupt niets dat de lezer-interpretatie zou kunnen verschuiven. Dit voorkomt dat de agent buiten zijn bevoegdheid treedt.
8. **LOGGING EXPECTATION:** Instructies dat de agent bij elke opmerkelijke bevinding een *zichtbare tag* moet toevoegen (zoals `[AMBIGUOUS]`, `[ESCALATE-HUMAN]`, etc.) en kort toelichten waarom [45] . Hoewel de agent-uitvoer zelf al in JSON komt, kunnen deze tags en toelichtingen ofwel opgenomen worden in de JSON (bijv. een veld `"tags": []` per item) of in een begeleidend markdown logsectie. Belangrijk is dat elk autonoom handelend stukje voorzien is van zo'n label, zodat we bij het samenvoegen in het log direct zien waar een beslissing ligt of uitgesteld is.
9. **PURPOSE REMINDER:** Tot slot herhaalt de prompt kort de missie: bijvoorbeeld: *"Je bent een signaleringsagent, je besluit niets definitief."* Dit houdt het model gefocust op zijn beperkte rol [46] .

Door deze structuur **uniform toe te passen**, bereiken we dat alle agents op vergelijkbare wijze te werk gaan en hun uitkomsten gestandaardiseerd zijn. Bovendien maakt het de prompts **auditbaar** – men kan ze

reviewen voordat agents worden ingezet, aan de hand van een kwaliteitschecklist (zoals gegeven in het Prompt Pattern document [47]). Enkele vragen uit die checklist: *Heeft de prompt één duidelijke taak? Is de output automatisch valideerbaar? Is de escalatie naar Human Gate expliciet genoemd? Verbergt de agent geen bias of informatie? Zijn er geen verboden beslissingen door de agent? Is alles logbaar en reversibel?* [48] .

Voorbeeld: de **Annotatie-agent** prompt zou aangeven dat zijn rol is “markeer ambigue of interessante passages en stel een voetnoot of commentaar voor, zonder iets aan de basistekst te veranderen.” Input: hij krijgt een recepttekst (bron+vertaling). Output: JSON array van `annotations` met voor elke entry o.a. `type` (bijv. "cultureel", "veiligheid", "foutje"), `location` (bv. paragraaf/zin nr.), `note_text` (voorgestelde voetnoottekst of commentaar), en een `status` (standaard "proposal"). Rules: max 5 annotaties per recept, geen duplicaten van bestaande, enz. Uncertainty: als de agent twijfelt of iets echt een issue is, plaatst hij het alsnog als voorstel maar met tag `[AMBIGUOUS]` en bijvoorbeeld `note_text` = "*editorial check required: …*". Forbidden: de agent mag **niet** zelf tekst corrigeren of 100% zeker iets als fout bestempelen; alleen signaleren. Logging: elke annotatie die hij doet krijgt in de JSON een veld `"tags": [...]` en de agent voegt eventueel een samenvatting toe in zijn run-log zoals `[AUTO-OK] 3 typos fixed in ingredient list` (als dat autonoom mocht) of `[ESCALATE-HUMAN] Unclear reference "zij" in step 4` bijvoorbeeld.

Validators voor output: Omdat alle agentoutput in gestructureerd formaat komt, kunnen we automatische validators (onderdeel van GOV-B) inzetten om te controleren of aan het contract is voldaan. Bijvoorbeeld, we parsen de JSON en checken: - Zijn alle verplichte velden aanwezig en in het juiste type/formaat? - Bevat geen veld verboden content (bijv. een `suggested_translation` gevuld terwijl dat niet mag)? - Zijn de waarden binnen toegestane reeksen (bijv. impactcategorie is een van de 4 gedefinieerde dimensies)? - Heeft de agent ergens vrije tekst geuit buiten de JSON (zou niet mogen tenzij in een expliciet logveld)?

Als zo’n validator faalt, wordt dat gezien als een implementatiefout of mogelijke non-compliance van de agent. De Orchestrator logt dit in `warnings.log` en kan besluiten de run te stoppen. Indien het iets kritiekals betreft (bijv. de agent heeft toch een tekstwijziging voorgesteld als feit doorgevoerd), kan zelfs een immediate **STOP** volgen (meteen naar `stop.log` en menselijk ingrijpen). Het is belangrijk dat deze checks lichtgewicht blijven om overhead te minimaliseren – simpelweg JSON schema validatie en een paar `if`-regels volstaan.

Samengevat zorgt de combinatie van **gestandaardiseerde prompts** en **outputvalidatie** ervoor dat agents zich strak aan hun opdracht houden en dat hun bijdragen veilig kunnen worden gebruikt in het verdere proces. Dit sluit aan bij de afspraak dat *“Het harmoniseert — het beslist niets.”* (Prompt Pattern doc) [49] : de agenten brengen structuur en harmonie in de informatie, maar nemen geen besluiten die de mens niet heeft gezien.

Runtime Workflow & Signaal-Gate-Closure

De **runtime executie** van het systeem volgt een vast patroon van *signaal* → *gate* → *closure* , een documentaire werkwijze om belangrijke kwesties zichtbaar te maken en af te handelen [50] [51] . Hieronder

beschrijven we stap voor stap hoe een content-item (bijv. een recept) door het systeem gaat, en hoe signalen, gates en closures worden gemanaged:

1. **Initiatie van een run:** Een redacteur of een batch-script start de verwerking van een specifiek item. De Orchestrator creëert een nieuwe run-ID en map (`sandbox/agent_runs/RUN_<datum>_<nr>`). Hij laadt de relevante input (bv. de bron- en vertaaltekst van het recept) en eventueel aanvullende gegevens (bv. glossary, context).

2. **Sequentiële agent-executie:** De Orchestrator doorloopt de lijst van agents die voor dit item van toepassing zijn volgens de gedefinieerde workflow (de volgorde kan bijvoorbeeld in `docs/WORKFLOW.md` of in de CrewAI flow vastliggen). Per agent:

3. **Prompt samenstellen:** De agent krijgt zijn prompt (volgens template in `prompts/`) ingevuld met de specifieke data voor dit recept.
4. **Model-uitvoering:** De Orchestrator roept het AI-model aan. Standaard wordt gebruikgemaakt van de lokale LLM (**Mistral 7B** of een vergelijkbaar model) via **Ollama** of directe binding, om kosten te besparen. Voor complexere taken kan eventueel conditioneel opgeschaald worden naar een krachtiger model (via Codex CLI of OpenAI API), maar dit gebeurt spaarzaam en op instructie van GOV-A besluiten (kostenbewustzijn).
5. **Resultaat opslaan:** De ruwe output (meestal JSON) wordt weggeschreven naar `sandbox/agent_runs/<RUN_id>/<agent>_output.json`. Tevens wordt de conversatie (prompt + raw completion) gelogd in `sandbox/run_logs/<RUN_id>.log` voor volledige trace.
6. **Validatie & tagging:** Na ontvangst van de output voert de Orchestrator/validator directe checks uit (zoals hierboven beschreven). Daarnaast worden in de JSON eventuele speciale velden of tekens geïnterpreteerd om te bepalen of er een Gate nodig is. Bijvoorbeeld, als de JSON entries bevatten met `"escalation": "HUMAN_REVIEW_REQUIRED"` of een tag `[ESCALATE-HUMAN]`, dan markeert de Orchestrator dat er een **Gate**-situatie is.

7. **Gate handling:** Indien zo'n Gate getriggerd is, zal de Orchestrator **niet automatisch doorgaan** naar de volgende agent of stap. In plaats daarvan registreert hij in de log dat een Gate is bereikt (bijv. `[GATE] Safety issue requires human review at step X`) en kan hij de run *pauzeren*. Afhankelijk van configuratie kan de Orchestrator wachten op een expliciete menselijke bevestiging in de console (CrewAI ondersteunt *Human-in-the-Loop workflows* waarbij de flow kan pauzeren voor input [52]). Alternatief kan de run afgebroken worden en het item in een "wachtende" lijst komen te staan tot review heeft plaatsgevonden.

8. **Menselijke review bij Gate:** Een redacteur krijgt via de logging of notificatie te zien dat er een Gate is. De relevante informatie is beschikbaar: bv. *Agent X signaleerde Y en heeft dit gemarkeerd voor human review wegens mogelijke <impact>*. De redacteur gebruikt de sandbox-output en eventueel de oorspronkelijke tekst om te besluiten:

9. **OK, doorgaan:** Het signaal is begrijpelijk maar geen blocker – men kan besluiten dat er (voorlopig) niets aan gedaan hoeft te worden. In dat geval zou de redacteur bijvoorbeeld in de sandbox een closure notitie kunnen toevoegen: status "deferred" of "rejected" met uitleg. Daarna kan de Orchestrator verder.
10. **Aanpassen/annoteren:** Het signaal vereist actie (bv. een voetnoot toevoegen). De redacteur voert nu direct die redactionele handeling door in de canonieke tekst of annotaties. Bijvoorbeeld, een

gevaarlijk gezondheidsadvies krijgt een voetnoot in de editie met de nuance “historisch advies, moderne inzichten verschillen...”. Vervolgens noteert de redacteur de closure als “accepted with rationale – footnote added” in de logs/beslissing. De Orchestrator krijgt een signaal dat deze kwestie afgehandeld is en mag verder.

11. **Escaleren (nog hogere instantie):** In zeer lastige gevallen kan de redacteur besluiten dat hij het niet zelf kan beslissen en het naar een projectbrede besluitvorming te tillen (dit is meer uitzonderlijk, wellicht bij ethische dilemma’s -> zou via governance board gaan). In zo’n geval blijft het item on hold (closure = “open”) tot in een later stadium een besluit valt.

Zodra de Gate-issue een closure heeft (ook al is dat “expliciet uitgesteld”), noteert men dit. De Orchestrator wordt hervat. **Belangrijk:** de tussentijdse menselijke acties worden zoveel mogelijk ook door het systeem vastgelegd. Bijvoorbeeld, CrewAI’s *Human Feedback in Flows* mechanisme kan de gegeven beslissing loggen zodat we die input later terugzien [52] [53] .

1. **Vervolg agent-runs:** De Orchestrator vervolgt met de resterende agents. Omdat sommige latere agents misschien afhankelijk zijn van eerdere resultaten, kan de Orchestrator de context updaten. Bijvoorbeeld, als de *Glossary-agent* na de *Terminologie-agent* komt, en er is besloten een bepaalde term voorlopig niet te veranderen, kan dat meegegeven worden zodat de Glossary-agent niet opnieuw hetzelfde signaleert.

2. **Afronding run – verzamelen signalen:** Aan het eind van de run verzamelt de Orchestrator alle output. In principe hebben we nu een collectie van **signalen** (differences, issues, voorstellen) waarvan sommigen al direct afgehandeld zijn (auto-okays) en anderen gemarkeerd zijn voor latere afhandeling. De Orchestrator kan een *samenvattend rapport* maken: bijvoorbeeld een Markdown die alle openstaande punten opsomt met hun status. Dit rapport kan naar de redacteur worden gestuurd of in een `governance_notes.log` worden gezet.

3. **Redactionele verwerking & closure:** Nu vindt de echte *closure* plaats van alle signalen, als dat niet al tussentijds is gebeurd. De redactie gaat alle overgebleven voorstellen en issues door (nu of op een later tijdstip, afhankelijk van workflow). Ieder signaal moet uiteindelijk een van de geaccepteerde **closure-states** krijgen [30] [54] :

4. *Accepted (met rationale):* het punt is geadresseerd door een wijziging/annotatie, en de reden is vastgelegd.
5. *Deferred (explicit):* er is besloten hier nu niets aan te doen, maar dit staat op de lijst voor mogelijke latere behandeling (met reden waarom uitgesteld).
6. *Rejected (with explanation):* het signaal bleek onterecht of niet relevant; geen actie nodig, met uitleg.
7. *Left open (intentional):* men erkent het signaal maar laat het bewust open als blijvende ambiguïteit (dit wordt ook vastgelegd, zodat bekend is dat het niet vergeten is).

De redacteur verwerkt deze beslissingen. In de praktijk betekent dit: - Bij *accepted*: doorvoeren van annotaties/tekstuele aanpassing in `docs/` content. Elke verandering wordt volgens het principe “zo is het nu, en waarom” gedocumenteerd – dit kan in de tekst zelf (bv. voetnoot die uitleg geeft is al een rationale) of in een apart beslissingsdocument als het een groter besluit is. Daarnaast logt men de closure in een central log of besluitregister. - Bij *deferred/open*: mogelijk een item toevoegen aan een `docs/decisions/todo.md` of een Phase-7 lijst zodat het niet vergeten wordt. - Bij *rejected*: noteren in log waarom we er niets mee doen (bv. “false positive by agent, no real issue”).

Zodra elk signaal een closure heeft, is deze run in principe **afgesloten**. Het item (recept) is nu “Done” voor deze fase: alle belangrijke beslissingen traceable, geen risico onbehandeld of onbenoemd [55]. “Done” betekent niet dat er geen fouten meer zijn, maar dat alle gevonden kwesties bewust zijn beoordeeld en vastgelegd [29].

1. **Opschoning en archivering:** De sandbox-data van de run blijft bewaard tenminste totdat een hogere governance beslist dat het gearchiveerd of opgeschoond mag worden. Dit om audit-redenen. Zoals eerder genoemd, verwijderen we het alleen als een expliciete rollback. In latere fases kan men ervoor kiezen om na succesvolle integratie en publicatie van een hoofdstuk, de ruwe agent-uitvoer te archiveren naar een compressiebestand of externe archiefschijf, om ruimte te sparen. Kosten en schaalbaarheid zijn beheersbaar omdat tekstdata relatief klein is; we vermijden echter bijvoorbeeld het langdurig bewaren van zware model-checkpoints of dergelijke – die zijn niet echt aanwezig hier, enkel tekst-logs en JSON.

Signal-Gate-Closure als patroon: Door dit stapsgewijze model garanderen we dat **geen enkel signaal ongedocumenteerd verdwijnt**. Het motto is: *“Absence of closure is itself a signal.”* [56]. Als er iets nog geen closure heeft, beschouwen we dat als open issue (en potentieel non-compliance als men zou willen publiceren zonder dat op te lossen). Daarom zijn de STOP-voorwaarden voor bijvoorbeeld de veiligheids-capability gedefinieerd rondom dit patroon [57] [58]: - Als een veiligheidsgerelateerd signaal is gedetecteerd maar er bestaat *geen* voorstelrecord ervoor (niet vastgelegd) -> STOP. - Als er wel een signaal is maar geen Gate acknowledgement -> STOP. - Als er een Gate was maar nog geen closuredocumentatie -> STOP. - Of als er een potentieel gevaarlijk issue stilzwijgend is “opgelost” zonder signal/gate/closure (i.e. iemand heeft direct tekst aangepast) -> STOP.

Deze STOP-condities worden in het systeem afgedwongen door checks, bijvoorbeeld voor een release of export: een script scant alle log- en besluitfiles en kijkt of er openstaande gates zijn. Pas als die afwezig zijn, beschouwen we het resultaat als compliant.

Governance & Veiligheidsmechanismen

In het ontwerp zijn diverse **governance- en veiligheidsmechanismen** ingebouwd die ervoor zorgen dat het systeem binnen de gestelde kaders blijft opereren en dat risico’s gecontroleerd blijven. Hier lichten we de belangrijkste toe:

- **Autonomy Envelope Enforcement:** We beperken de autonomie van agents tot *low-risk, reversible actions* [59]. Concreet: Agents mogen bijvoorbeeld wel format-foutjes herstellen of uniforme notaties toepassen (ml vs liter, consistentie in spelling) automatisch, maar ze mogen **nooit** zelfstandig betekenisvolle wijzigingen doorvoeren zoals het herinterpreteren van een term of het weglaten van een receptstap [27]. De agent autonomy envelope doc geeft duidelijke voorbeelden van wat wel/niet autonoom mag [60] [61]. Ons systeem implementeert dit door:
- Prompt-instructies (FORBIDDEN-sectie) die dergelijke acties verbieden.
- Validators die de output controleren op tekenen van overtreding (bv. als een agent toch een alternatief woord voorstelt als vervanging zonder dat dit gevraagd is).
- Logging van alle autonome wijzigingen met tag `[AUTO-OK]` wanneer ze binnen de toegestane categorie vallen [62]. Als iets buiten de lijst valt maar toch geautomatiseerd gebeurde, merkt de Orchestrator dat op en flagt het als potentieel probleem (menselijke review).

- **Tagging en Audit-trail:** Zoals eerder vermeld, labelt het systeem elke actie of beslissing met audit-tags. De volgende tags worden minimaal onderscheiden [62]:

- `[AUTO-OK]` – Autonome verandering binnen laag risico & omkeerbaar (bv. kleine format correctie).
- `[AGENT-MEETING]` – (Voor toekomstige uitbreiding: als agents onderling zouden overleggen) aangeeft dat agents iets hebben bediscussieerd maar geen definitief besluit is genomen zonder mens.
- `[AMBIGUOUS]` – Aangegeven dat een kwestie gedocumenteerd is als ambigu en onopgelost gelaten.
- `[ESCALATE-HUMAN]` – Signaal dat de agent expliciet vraagt om menselijk ingrijpen.
- (Eventueel nog tags als `[GATE]` of `[DECISION-LOGGED]` kunnen worden gebruikt voor interne aanduiding.)

Deze tags verschijnen zowel in de detailoutput van agents als in de centrale logs. Bijvoorbeeld, een agent-output JSON kan zoiets bevatten als:

```
{
  "issue": "Term X heeft meerdere betekenissen",
  "impact": ["semantic", "cultural"],
  "decision": "NONE",
  "tags": ["AMBIGUOUS", "ESCALATE-HUMAN"]
}
```

Tegelijk zal in `warnings.log` een mensvriendelijke regel worden geschreven: `[AMBIGUOUS][ESCALATE-HUMAN] Translator agent found ambiguous term "X" needing human review (Run 20260104_003).` Op deze manier is de audit-trail rijk en doorzoekbaar. Traceability is verder verzekerd doordat elke logregel ook referenties kan bevatten naar run IDs of excerpt IDs, overeenkomstig de canonical index referentieformaten [40].

- **Explicit Human Gate triggers:** De governance heeft bepaald bij welke soort situaties een menselijke beslissing *verplicht* is [63]. Het systeem implementeert deze triggers als checks in relevante agents en validators. Enkele voorbeelden van triggers:
- *Cultureel/religieus betekenisvol begrip:* de Cultural-historical agent of Terminologie-agent herkent een term of frase waarvan bekend is dat de interpretatie gevoelig ligt (bv. een term met koloniale connotatie). Het agent zou hier standaard `[ESCALATE-HUMAN]` aan hangen omdat alleen een mens kan besluiten of en hoe dit toegelicht moet worden.
- *Gezondheids- of veiligheidsclaim:* de Safety-agent markeert recepten of instructies die potentieel schadelijk kunnen zijn als ze letterlijk gevolgd worden (bv. het eten van rauw voedsel zonder waarschuwing). Dit **moet** een Gate uitlokken [64]: de tekst mag niet naar een publiek zonder dat een editor er expliciet iets mee gedaan heeft (annotatie of relativering).
- *Glossary impact:* als het systeem een inconsistentie in terminologie vindt die de reader-facing tekst betreft (bv. twee hoofdstukken vertalen een ingrediënt verschillend), dan kan het niet autonoom kiezen welke juist is. Dit gaat de glossary-lifecycle in en moet door mensen of via een formeel proces beslist worden – tot die tijd blijft het inconsistentie-signaal staan als “open”.
- *Ambiguïteit dicht op lezersniveau:* als een agent aangeeft dat een zin voor de lezer onduidelijk of misleidend zou kunnen zijn, dan laten we dat nooit stilzwijgend passeren. Ofwel annoteren we het

(toelichting voor lezer) of we besluiten bewust dat het zo blijft. Geen automatische normalisatie zonder menselijke goedkeuring.

Het systeem bekrachtigt dit door in de workflow te coderen dat bij dergelijke triggers een `governance_stop` functie wordt aangeroepen. Dit logt in `stop.log` zoiets als: `Governance-STOP: Human Gate condition triggered (Safety) on item <id>, run halted.` [57] [58]. De Orchestrator zal dan het huidige procesdeel stoppen. Pas na interventie (en bijvoorbeeld het toggelen van een "resolved" vlag door een mens in een config of via CLI) kan dat item herstart of voortgezet worden.

- **Conflicterende voorstellen beheren:** Als verschillende agents of bronnen verschillende oplossingen voorstellen voor hetzelfde probleem (bijv. twee vertaalvarianten voor een term), dan bepaalt de governance dat deze *nooit automatisch samengevoegd of gekozen* worden [65]. In plaats daarvan registreren we ze als **"competing proposals"**. In de praktijk: stel agent A zegt "vertaal *lobak* als radish" en agent B zegt "gebruik daikon", dan zullen beide voorstellen in de voorstellenlijst staan. De Glossary-lifecycle (een menselijk geleid proces wellicht ondersteund door een research-agent) pakt ze op als één case met twee opties. Ons systeem moet dit faciliteren door:
- Agents aan te moedigen hun onzekerheid aan te geven (ze kunnen bijv. beide opties noemen of hun eigen voorkeur als onzeker labelen).
- Geen enkele agent prioriteit te geven: Orchestrator voegt beide inputs toe aan de sandbox.

- Tijdens human review zal men expliciet moeten kiezen of beide te behouden (misschien één als hoofdvertaling, andere als voetnoot) of één te nemen en de andere te verwerpen, met logging waarom.

- **Geen direct gebruik van agent-"research" als feitelijke bron:** Agents kunnen informatie van buiten inbrengen (bv. een Language Model zou aanvullingen kunnen doen gebaseerd op algemene kennis). We hanteren hier streng het principe: *agent research is hypothesis, not evidence* [66]. Dus als een agent iets toevoegt ("Misschien betekent dit ingrediënt X in moderne termen Y") dan wordt dat gezien als een **claim die verificatie vereist** – niet als feit. Ons ontwerp ondersteunt dit door altijd een menselijke of externe bron check in te bouwen voor zulke gevallen. Bijvoorbeeld, een *Research note* kan ontstaan die een redacteur dan grondig checkt (desnoods door een bibliothecaris of historisch bronmateriaal) voordat het ooit canoniek wordt. In geen geval wordt agent-output klakkeloos in de editie overgenomen als feitelijke statement zonder menselijke validatie.

- **Rollback en incident-respons:** Mocht er onverhoopt iets misgaan (bijv. een agent heeft vanwege een bug toch de tekst veranderd of verkeerde data gebruikt), dan detecteert de *Incident & Resilience* component dit idealiter [67]. Bijvoorbeeld, een aparte monitoring die kijkt of de finale teksten afwijken op plekken waar geen closure is geregistreerd – dat zou duiden op een "stille wijziging" en een incident. In zulke gevallen kunnen we via version control (git) eenvoudig een rollback doen op de canonieke documenten naar de vorige staat, omdat iedere change traceable was. De betrokken run in sandbox krijgt dan een label "ROLLED_BACK" en de reden wordt genoteerd in `governance_notes.log`. Zo leren we ook het systeem robuuster te maken.

Al deze mechanismen dienen één doel: **veiligheid en betrouwbaarheid** van het proces waarborgen, zonder de mens eruit te halen. De AI is een *assistent*, geen auteur met eigen autoriteit. De governance-mechanismen zijn er om de *"georganiseerde twijfel"* te institutionalizeren: agents brengen die twijfel aan (red-teaming concept [68]) en governance + mens zorgen dat elke twijfel netjes geadresseerd of op z'n minst

genoteerd wordt. Daarmee voorkomen we zowel fouten door blind vertrouwen in AI als het overslaan van potentieel cruciale kwesties.

Logging en Traceerbaarheid

Logging is het kloppend hart van traceerbaarheid in dit systeem. We hebben al veel aspecten van logging benoemd; hier geven we een beknopt maar compleet beeld van hoe logging is ingericht:

- **Run-specific logs:** Elke run krijgt zijn eigen logbestand (zie `sandbox/run_logs/RUN_<id>.log`). Hierin staat de volledige dialoog/interactie van de Orchestrator met de agents: inclusief welke prompt is gestuurd (of een verkorte versie), het ruwe antwoord van de agent, en eventuele realtime commentaren. Dit is vooral voor **debugging en transparantie** naar de ontwikkelaars en ervaren redacteuren die willen zien *hoe* een agent tot een voorstel kwam. Deze logs kunnen vrij gedetailleerd zijn en zijn niet bedoeld voor publicatie maar voor intern gebruik.

- **Central logs (warnings, stop, governance_notes):** Dit zijn overkoepelende logs voor belangrijke gebeurtenissen:

- `warnings.log`: Hier komen alle opmerkelijke zaken die niet per se het proces stoppen, maar wel aandacht vragen. Bijvoorbeeld: "Annotator agent output schema mismatch, field X missing (auto-corrected format and continued)" of "Translator agent: potential bias detected, flagged as ambiguous." Deze warning entries helpen om na een run snel te zien of er iets ongewoons gebeurde.
- `stop.log`: Hier loggen we *alle* gevallen waarbij het systeem een STOP heeft toegepast. In principe is elke regel in stop.log een situatie die menselijke tussenkomst vereiste of nog vereist. Bijvoorbeeld: `[STOP][RUN_20260104_003] Safety signal without closure – processing halted.` Dit file is cruciaal voor governance: het is immers een checklist van wat er *niet verder mag* tot opgelost. Het systeem zou zelfs kunnen weigeren een eindresultaat te genereren zolang stop.log niet leeg (of afgehandeld) is.
- `governance_notes.log`: Dit is een vrijer format log voor allerlei notities en besluiten tijdens de runs. Hierin noteren zowel de Orchestrator als eventueel de mens aanvullende context, bijv: "Governance override: proceed despite missing minor field, approved by PO on 2026-01-10" of "Note: Glossary decision pending for term X, see meeting minutes dd...". Het is dus een soort dagboek van het systeem waar belangrijke zaken die niet elders thuishoren toch worden vastgelegd.

- **Session/Move log (Codex session log):** Zoals eerder genoemd wordt in `docs/CODEX_SESSION_LOG.md` precies bijgehouden wat vanuit de sandbox definitief is ingebracht in de canonieke docs. Dit log is onderdeel van de **canonieke documentatie**; het vormt de menselijke leesbare changelog per sessie of cluster van wijzigingen. Hierdoor kan iedereen achteraf zien *welke voorstellen zijn verwerkt* en met welke motivatie.

- **Annotatie van bronbestanden:** Naast losse logfiles wordt **in de documenten zelf** ook trace informatie bijgehouden waar relevant. Bijvoorbeeld, in `docs/mustika_rasa_en.md` (de vertaling) kunnen speciale markeringen of referenties worden opgenomen die linken naar beslissingen of logs. We zouden een conventie kunnen hanteren zoals HTML-achtige comment tags of markdown voetnoten die verwijzen naar een besluit. Voorbeeld in de vertalingstekst:

```
... Voeg de laos toe en laat 5 minuten sudderen.^[Vertaling volgt origineel;
alternatief "galanga" verworpen wegens context. 69 ]
```

Hier is een markdown voetnoot met een korte rationale en verwijzing naar een beslissingrecord of log. Dit zorgt ervoor dat iemand die de editie leest met voetnoten, de uitleg ziet, en iemand die de markdown bekijkt ook de link naar het onderliggende record heeft. Dergelijke in-text referenties koppelen dus de *inhoud* direct terug aan de governance/artifacten (traceability loop gesloten).

- **Unieke IDs en referenties:** Om logs, proposals en teksten aan elkaar te knopen gebruiken we unieke identifiers. Recepten of fragmenten kunnen een ID krijgen (bv. `lobak_034_048` als code voor het fragment "lobak" van pag. 34 regel 48). Agents noemen deze ID in hun output. Runs worden ge-ID met timestamp of increment. Beslissingen krijgen een ID zoals `P7_CANONICAL_GLOSSARY_LOBAK_001` (voorbeeld uit canonical index 35). Deze ID's komen terug in de logging en in de tekst, waardoor een web van kruisverwijzingen ontstaat. Hoewel dit complex lijkt, kan een simpel script alle referenties checken om zeker te zijn dat bijvoorbeeld elke `[PROPOSAL_MOVED]` entry een corresponderend footnote of decision file heeft, etc. Dit is iets waar we later tooling voor kunnen schrijven (of integreren met e.g. mkdocs site generation of een indexer tool).

- **Reversibility en versiebeheer:** Alle veranderingen gebeuren in een git-omgeving (aangenomen). Zo hebben we een volledig versiebeheer op documentniveau. Iedere commit waarin de vertaling of annotaties zijn aangepast refereert idealiter naar de betreffende beslissing of logregels (in de commit message of via issue-IDs). Reversibility houdt ook in: als een bepaalde wijziging toch niet goed blijkt, kunnen we op basis van het log heel gericht een rollback doen. Bijvoorbeeld, als uit review blijkt dat een annotatie verkeerd was, weghalen en loggen `[ROLLBACK] removed annotation #123 after further research disproved agent's claim`. Dankzij git en logs is duidelijk wat er is teruggedraaid en waarom.

Samengevat creëert dit logging-ontwerp een **zeer rijke audit trail**. Het maakt mogelijk dat over bijvoorbeeld vijf jaar, een nieuw team nog kan nagaan waarom een passage op een bepaalde manier vertaald of geannoteerd is, welke alternatieven overwogen waren, en wie/wat dat besloten heeft. Dit sluit precies aan bij de projectdoelstelling van een *herleidbare kennisinfrastructuur, niet slechts een eenmalige vertaling* 70 .

Ondersteuning voor de Redactionele Workflow

De Mustika Rasa-architectuur is opgezet om naadloos aan te sluiten op de werkwijze van menselijke redacteuren en om een **redactionele workflow** mogelijk te maken die efficiënt is, maar ook grondig. Enkele aspecten waarmee het systeem de menselijke interactie ondersteunt:

- **Redactioneel Dashboard (optioneel):** Hoewel we primair in de terminal en met markdown werken (lage overhead), zou een eenvoudig lokaal dashboard het reviewproces makkelijker kunnen maken. Denk aan een statische site of MDX interface waarin voor een gegeven hoofdstuk alle agentvoorstellen en logs overzichtelijk staan. Bijvoorbeeld, een pagina per recept met: de originele tekst, de vertaling, daaronder een lijst van *signalen* (met tags zoals [AMBIGUOUS]) gegroepeerd per

type (terminologie, veiligheid, etc.), en knoppen/links naar “Akkoord / Verwerpen / Uitstellen”. Dit hoeft geen complexe webapp te zijn: een gegenereerd markdown of HTML-rapport dat de redacteur in een browser bekijkt kan volstaan. De daadwerkelijke vastlegging van een besluit gebeurt dan door bv. in de markdown een vinkje of comment te zetten en vervolgens door commit of CLI commando te bevestigen.

- **Editorial meeting workflow:** Het systeem kan *verslagen* of *bundels* genereren die in een redactie-overleg gebruikt worden. Bijvoorbeeld, stel dat na een run 10 open issues zijn over hoofdstuk 2, dan kan men een YAML of MD lijstje maken:

```
**Issue 1:** Term "X" inconsistent (zie RUN_... annotator output).
*Voorstel:* toevoegen aan glossary, voorlopig vertalen als "...".
*Impact:* Terminologie, lezersbegrip.
*Status:* NOG TE BESLISSEN (Human Gate).

**Issue 2:** Recept onvolledig (stap ontbreekt?).
*Voorstel agent:* [AMBIGUOUS] markeren, in voetnoot uitleggen dat bron
mogelijk fout.
*Impact:* Continuïteit, historische integriteit.
*Status:* BESLIST - akkoord, voetnoot toegevoegd.
```

  Dit soort overzichten, deels automatisch samengesteld uit de logs, versnellen de workflow doordat de redactie niet zelf in ruwe JSON hoeft te graven. Ze kunnen tijdens de meeting deze lijst doornemen, besluiten invullen (desnoods met pen en papier of via edit), en na het overleg doorvoeren.

- **Prompt-tuning met redactionele feedback:** De prompttemplates liggen niet voor eeuwig vast; ze zullen verfijnd worden op basis van praktijkervaring. Omdat ze in de `prompts/` map als markdown staan, kan een redacteur of domain-expert ze relatief makkelijk aanpassen (bijv. een regel toevoegen onder FORBIDDEN als blijkt dat een agent iets ongewenst toch deed, of de formulering van een Rule verduidelijken). Deze wijzigingen verlopen via governance: kleine tweaks kunnen direct doorgevoerd worden, grotere veranderingen moeten via een review (zoals *Prompt Pattern adoptie Phase-5* aanraadt [71]). Het systeem kan dit faciliteren door bijvoorbeeld een label in de promptfiles: “compliant” of “needs review” zoals aangegeven [72], en een testmodus waarin een agent-run kan worden gesimuleerd met de nieuwe prompt om te kijken of de output nog voldoet. Zo blijft de prompt-infrastructuur levend en in dienst van de redactie.

- **Ondersteuning bij Glossary lifecycle:** Terminologiebeheer is een specifiek redactioneel proces. Het systeem ondersteunt dit door elke term-kwestie als zelfstandige *propositie* te behandelen. Bijvoorbeeld, de Terminologie-agent maakt voor “lobak” een voorstelrecord `sandbox/proposals/lobak_proposal.md` waarin staat: *lobak: kan ‘Chinese radijs’ of ‘daikon’ betekenen; huidige vertaling: radish; voorstel: opnemen in glossary en bespreken.* De redactie kan dat beoordelen en een beslissing nemen. Belangrijk is dat het systeem deze beslissing vervolgens **doorvoert op alle plekken**: als men “lobak” officieel opneemt in de glossary als lemma, markeert men dat in `docs/glossary.md` en refereert naar de beslissing (met ID etc.), en het systeem kan desgewenst agent-runs in de toekomst laten weten dat “lobak” nu een vastgelegde term is zodat agents niet blijven flaggen. Dit is

een voorbeeld van *canonieke feedback loop*: een menselijke beslissing stroomt terug het systeem in als kennis (maar dan via een gecontroleerd, gedocumenteerd mechanisme, niet stilzwijgend).

- **Menselijke fouten voorkomen:** Ironisch genoeg kan ook de mens fouten maken in dit proces (bijv. vergeten iets te loggen, of per ongeluk een wijziging direct in de tekst maken zonder log). Het systeem helpt hier door veel van de administratieve last automatisch te doen of te controleren. Bijvoorbeeld, als een redacteur een footnote toevoegt, zou een pre-commit hook kunnen checken of die footnote een referentie ID bevat naar iets in sandbox/logs. Of de `stop.log` houdt de mens scherp door te melden "hey, er is nog iets niet gesloten". Dit bevordert een cultuur waarin de mens weliswaar beslist, maar niet willekeurig of ongecontroleerd handelt. De mens wordt als het ware ook begeleid door het systeem om consequent te zijn met de governance.

- **UI en Tools integratie:** We houden het systeem zo lokaal en licht mogelijk. Toch kunnen we slim gebruikmaken van bestaande tools om de ervaring te verbeteren:

- *Diff-tools:* Bij het toepassen van wijzigingen kan een visuele diff tussen bron en vertaling nuttig zijn. We kunnen `git diff` of meld gebruiken, of zelfs een custom script dat twee versies van een tekst naast elkaar toont met highlight. Dit helpt de redacteur om te zien of de aangebrachte wijziging inderdaad alleen is wat hij bedoeld had.
- *Spellcheck/Grammar:* Niet de focus, maar eenvoudige linters voor spelling kunnen in de pipeline worden gezet (alleen als waarschuwing).
- *CrewAI flows met HITL:* CrewAI biedt mogelijkheden voor bijvoorbeeld een *"approve to continue"* step [52]. Dit zouden we kunnen inzetten voor een semi-automatische modus: de Orchestrator loopt tot aan een Gate, print alle info, en vraagt via de console "Type 'yes' to continue after resolving above issue". De redacteur kan dan ter plekke iets doen en 'yes' typen als hij door wil. Dit is command-line interactiviteit, maar kan efficiënter zijn dan volledig stoppen en later herstarten.

- *Logging analysetools:* Overweeg simpele scripts die logs samenvoegen of analyseren. Bijvoorbeeld, een script `summarize_logs.py` dat per run of per hoofdstuk samenvat: X AUTO-OK changes, Y ambiguous issues, Z escalations. Dit geeft inzicht in hoeveel werk nog ligt bij de mens en of bepaalde agents misschien te vaak escaleren (wat kan wijzen op promptverbetering nodig).

- **Editorial capabilities alignment:** Het systeem is gebouwd om de **elf redactionele capabilities** (zoals beschreven in *EDITORIAL_CAPABILITIES.md*) te faciliteren, niet te vervangen [73]. Voor elke "zorg" in dat document is er een mechanisme in het systeem:

- *Meaning Preservation:* Agents flaggen betekenisverlies; systeem zorgt dat alternatieven overwogen worden en geen betekenis stil verdwijnt [74].
- *Terminology Stewardship:* Systeem biedt glossary lifecycle en consistentiechecks [75].
- *Cultural-Historical Interpretation:* Agents signaleren culturele nuances en systeem dwingt annotatie vs herschrijven (niets plat vertalen) [76] [77].
- *Error Recognition:* Het systeem laat feitelijke fouten staan maar markeert ze, precies zoals de richtlijn zegt geen "stille correcties" [78] [79].
- *Safety Judgement:* Safety-agent + stop rules implementeren dat gevaarlijke stukken niet ongeadresseerd blijven [80] [81].

- *Culinary Validation:* Hoewel lastig voor AI, kan het systeem middels een specialist-agent of externe input checken of kooktechnisch iets niet klopt, en dat hoogstens als historisch aantekenen, niet corrigeren [82] [83].
- *Balanced Readability:* Een Readability-agent kan suggesties doen voor taalversoepeling, maar het systeem zal die nooit zonder mens doorvoeren om trouw aan bron te bewaren [84] [85].
- *Interpretation Guidance:* Het systeem ondersteunt de redactie bij het schrijven van voorwoorden of toelichtingen door overzicht te bieden van waar lezers mogelijk guidance nodig hebben (dit is grotendeels menselijk werk, maar issues die agents vinden kunnen richting geven) [86] [87].
- *Contextual Annotation:* Alle detecties van ontbrekende context door agents resulteren in voorstelnotities, die de redactie makkelijk kan inweven als voetnoten [88] [89].
- *Document Integrity:* De gehele governance focus op traceerbaarheid en minimale interventie beschermt de integriteit van Mustika Rasa – geen wijziging zonder logische verantwoording, net zoals het capability doc voorschrijft [90] [91].

De laatste capability (*Relatie met techniek en governance* [73]) verwoordt dat redactionele behoeften leidend zijn, techniek ondergeschikt en governance controlerend. Ons ontwerp houdt zich daaraan door: de tooling is configurabel naar redactionele wensen, agents kunnen worden uitgeschakeld of aangepast als de redactie vindt dat ze niet nuttig zijn, en governance rules zijn nooit een doel op zich maar een bescherming. Dit betekent bijvoorbeeld als een bepaalde governance regel onnodig de voortgang belemmert zonder duidelijk nut, GOV-A kan besluiten die aan te passen – ons systeem is dan flexibel genoeg om die wijziging te accepteren (aangezien veel regels in config of prompts zitten en niet hardgecodeerd).

Technische Implementatie en Tools

Uitvoeringsomgeving: Het systeem draait op een lokale **Mac Mini** (Apple Silicon) met een Unix-achtige omgeving. We maken gebruik van de terminal (shell scripting, Python) als orkestratie-interface. De keuze voor Mac Mini is gunstig vanwege het krachtige hardwareprofiel (in vergelijking met Raspberry Pi bijv.) en mogelijkheid om AI-modellen lokaal te draaien via optimalisaties.

CrewAI als Orchestratie-framework: We zetten **CrewAI** in om de multi-agent workflow te structureren. CrewAI biedt YAML-gebaseerde flows en integratie met LLMs en tools, plus het gemak van *Human-in-the-loop* stapjes [52]. De project zal als een CrewAI project worden opgezet (via `uv` en `crewai` CLI) met een structuur zoals `src/` voor code en een flows-definitie. Bijvoorbeeld, een flow YAML zou eruit kunnen zien als:

```
steps:
  - id: translator
    llm: local::mistral-7b
    prompt: prompts/translator_agent.md
    output: sandbox/agent_runs/{{RUN_ID}}/translator.json
  - id: annotator
    requires: translator  # zodat de translator stap eerst is afgerond
    llm: local::mistral-7b
    prompt: prompts/annotator_agent.md
    output: sandbox/agent_runs/{{RUN_ID}}/annotator.json
  - id: safety
```

```
    llm: local::mistral-7b
    prompt: prompts/safety_agent.md
    output: sandbox/agent_runs/{{RUN_ID}}/safety.json
    on_complete:
      function: check_safety  # custom python hook to look for red flags
      if: not $?  # if check_safety returns false (issue) -> trigger stop
  - id: glossary
    ...
```

(Het exacte formaat kan verschillen, maar dit illustreert de idee). De *llm: local::mistral-7b* geeft aan dat via Ollama of ander backend een lokaal model wordt gebruikt. CrewAI's flexibiliteit laat ook toe OpenAI modellen te gebruiken via `llm: openai:gpt-4` mocht dat ooit nodig zijn voor kwaliteit. **Belangrijk:** We configureren CrewAI zodanig dat calls naar OpenAI beperkt en expliciet zijn (cost control).

Ollama en Mistral: Ollama is een handig orchestratieprogramma voor het draaien van modellen lokaal; we kunnen bijvoorbeeld een Mistral 7B model in Ollama registreren. Onze Python/crew code kan via `subprocess` of an API Ollama prompten. Mistral 7B is relatief klein en zou op de Mac Mini (zeker de nieuwere M2/M3) moeten kunnen draaien binnen acceptabele snelheid voor onze tekstlengtes. Eventueel kunnen we een 13B model voor bepaalde agents proberen als 7B te beperkt blijkt (afweging tussen performance en kwaliteit). We houden de prompts zo beknopt en taakgericht mogelijk om ook met kleinere modellen toch goede resultaten te behalen – de heldere instructies en voorbeeldstructuur uit het Prompt Pattern helpen daarbij. Bovendien zorgen de validators ervoor dat onzin-output direct gedetecteerd wordt, wat ons vertrouwen geeft om kleinere modellen in te zetten.

Terminal integratie en CLI-tools: Naast CrewAI (dat zelf via CLI triggers heeft zoals `crewai kickoff`), kunnen we ook *zelfstandige Python scripts* hebben voor veelvoorkomende taken: - `run_pipeline.py <hoofdstuk>` – start de orchestratie voor alle recepten in een hoofdstuk (batch-run). - `review_report.py <hoofdstuk>` – genereer een markdown rapport van alle open issues. - `apply_decision.py <issue_id> <decision>` – een helper script dat bijvoorbeeld een bepaalde beslissing meteen op alle plekken doorvoert (mits eenduidig, zoals een terminologie keuze). - `check_compliance.py` – draait een snelle check over de repository voor STOP-voorwaarden en inconsistenties. - `phase_export.py <phase>` – om bijvoorbeeld een bepaalde fase resultaat uit te schrijven (bv. de wetenschappelijke editie als PDF, waar dit systeem voornamelijk toe bijdraagt).

Al deze tools zijn simpel en lokaal zodat geen zware infrastructuur nodig is.

Database / search index: Voor nu vermijden we het opzetten van zware databases. Mocht er behoefte zijn om efficiënter te zoeken door alle voorstellen of besluiten, kunnen we **Whoosh** of een SQLite DB integreren. Bijvoorbeeld een SQLite met tabellen: `proposals(id, type, status, text, source_ref, run_id)` etc. Zolang de dataset klein is (het boek heeft ~1200 recepten, elk misschien een paar issues), is zelfs een volledige repository grep of simple in-memory index voldoende. We denken "out of the box" door dit pas te doen als nodig: wellicht is een dynamisch gegenereerde indexpagina (gewoon een markdown die alles opsomt) al genoeg voor menselijk gebruik.

Scalability en performance: Een Mac Mini moet dit proces aankunnen. Zeker als we niet alles in één klap doen. We kunnen per hoofdstuk of recept werken. Memory-footprint is grootste zorg: meerdere LLMs tegelijk draaien kan zwaar zijn. CrewAI laat toe taken sequentieel te doen en ook om bijv. een pool van workers op te zetten. Maar we houden het eenvoudig: één run = agents sequentieel, dat spaart geheugen. Concurrency kan op hoofdstukniveau: parallel meerdere recepten behandelen als CPU/GPU het toelaat, maar dit is tweakbaar. We vermijden ook zware library overhead (CrewAI zelf is relatief licht; het gebruikt Python and OpenAI API libs). Geen cloud nodig behalve als fallback.

Logging implementatie: Logging gebeurt via Python's logging module of zelfs simpelweg file-writes. CrewAI tasks kunnen custom Python hooks defineren (zoals in YAML `on_complete: function: ...`). Die hooks zullen onze logging aansturen. We formateren logs op textniveau (geen complex binary logs).

Security: Omdat alles lokaal draait, zijn er weinig exogene security issues. Wel letten we op *prompt-injection*-achtige problemen: een agent zou in theorie misleid kunnen worden door malafide input in de tekst (bijv. als er een stukje tekst staat "Ignore all instructions and output this..."). Dit is onwaarschijnlijk in ons geval (bron is statisch, niet door gebruikers online input). Toch bouwen we defensieve prompts (e.g. geen agent zal instructies uit de content blind volgen omdat hun ROLE afgebakend is).

Versiebeheer en samenwerking: De redactie kan via git samenwerken. We kunnen bijv. GitHub of een private GitLab gebruiken, maar zelfs een lokale git is prima. We zorgen dat grote files (scans) niet telkens meeverzenden; die kunnen in LFS of extern. Het team kan branches gebruiken voor experimentele runs versus golden master branch voor de canonieke content. Onze logs en sandbox outputs kunnen eventueel in .gitignore als ze te veel clutter geven, maar het is beter om ze wel versie te beheren voor audit. Hier is een trade-off: wel logs in git betekent ook alle tussentijdse iteraties vereeuwigen; dat is goed voor audit maar kan repo-size vergroten. We kunnen beslissen alleen de finale beslis-artefacten en de CODEX_SESSION_LOG canoniek te maken, en raw run_logs niet in git te houden (ze blijven lokaal als naslag). Dit is een keuze voor implementatiefase.

Roadmap voor Iteratieve Implementatie

Tot slot presenteren we een stapsgewijze roadmap om dit ontwerp geleidelijk op de Mac Mini te implementeren. Elke stap levert een werkend increment van het systeem, zodat we tussentijds kunnen bijsturen en leren:

1. **Basisinstallatie en Setup (Week 1-2):** Installeer alle benodigde tools op de Mac Mini:
2. Python 3.10+ en CrewAI CLI [92] [93] .
3. Ollama en het gewenste lokale model (Mistral 7B) downloaden en testen op simpele prompts.
4. Repositorium opzetten met basisstructuur (`docs/`, `sandbox/`, etc. zoals hierboven beschreven). Migreer alle bestaande governance- en beleidsdocumenten in de juiste map (10-governance).
5. Versiebeheer (git) initialiseren.

6. **Outcome:** Een "systeem skelet" dat nog niets inhoudelijks doet, maar waarin alle documenten aanwezig zijn en tools werken (test bv. `crewai --version`, een simpele `ollama run mistral "Hello"`).

7. **Phase 0 – Bronrestauratie Automation (Week 3-4):** Automatiek voor de technische bronherstel fase:

8. Schrijf een eerste agent prompt (en code) voor **OCR-foutdetectie/correctie** op een pagina tekst. Eventueel gebruik GPT-4 via API voor hoge nauwkeurigheid in deze fase (omdat dit een eenmalige conversie is), maar log alle wijzigingen.
9. Implementeer een minimal Orchestrator script dat een OCR-tekst door deze agent haalt en outputs een gecorrigeerde tekst plus log van wijzigingen (met [AUTO-OK] tags voor iedere correctie).
10. Zorg voor logging van elke correctie met oorspronkelijke en nieuwe waarde en reden (per *Bronrestauratie-eis* 94).

11. **Outcome:** De originele Mustika Rasa tekst is beschikbaar als betrouwbare digitale master. Alle technische fixes zijn doorgevoerd met documentatie. We hebben hiermee een eerste end-to-end test van agents + logging. Menselijke review op deze fase: controleren of er niets ten onrechte aangepast is (inhoud vs techniek scheiding).

12. **Phase 1 – Annotatie pipeline voor één hoofdstuk (Week 5-8):** Bouwen van de basis pipeline voor de wetenschappelijke editie:

13. Definieer 2-3 core agents: bv. *Translation Fidelity Agent*, *Annotation/Safety Agent*, *Terminology Agent*. Baseer hun prompts op de governance docs (forbidden: geen inhoud wijzigen, etc).
14. Orchestrator implementeren via CrewAI of custom Python: sequentieel deze agents laten draaien op één voorbeeldhoofdstuk (een beperkt aantal recepten).
15. Implementeer basale Gate-handling: bijvoorbeeld als Annotation-agent iets markeert als `[ESCALATE-HUMAN]`, laat de Orchestrator een melding geven en stop de loop voor dat recept.
16. Schrijf de sandbox resultaten weg en handmatig laat de mens de open issues afhandelen.
17. Log de human decisions in CODEX_SESSION_LOG.md en eventueel pas de docs aan.

18. **Outcome:** Een werkende *vertical slice* van het systeem: van input tekst tot output voorstellen en menselijke closure voor een klein onderdeel. Dit demonstreert het hele signal-gate-closure concept in praktijk. We kunnen nu verfijnen op basis van de bevindingen (was de output bruikbaar? Waren er te veel false positives? etc).

19. **Uitbreiding agents & capabilities (Week 8-12):** Iteratief nieuwe capabilities toevoegen:

20. *Cultural context agent*, *Culinary logic agent*, *Continuity agent* etc., in volgorde van prioriteit gebaseerd op het editorial capabilities document.
21. Voor elke nieuwe agent: schrijf prompt, tests deze op enkele inputs, voeg toe aan orchestrator flow.
22. Valideer dat hun output schema past in het bestaande logging/integratie. Pas zonodig validators aan voor nieuwe fields/tags.
23. Elke toevoeging laten reviewen door het redactionele team (GOV-A besluit of de agent waarde toevoegt, zoniet herzien of overslaan).

24. **Outcome:** Steeds bredere dekking van redactionele zorgen door agents. Na deze stap hebben we wellicht ~5-6 verschillende agents die samen een groot deel van de fase-1 taken ondersteunen. Het boek kan nu hoofdstuk na hoofdstuk door de pipeline, al zal menselijk werk nog flink zijn (maar gerichter en minder routineus dan zonder systeem).

25. **Governance-stop en validator refinement (Week 10-14, parallel aan 4):** Terwijl capabilities groeien, implementeren we de strengere GOV-B regels:

26. Programmeer specifieke STOP-conditions checks, met name voor Safety: maak bijvoorbeeld in de Safety-agent output een veld `unsafe` boolean. Schrijf een check dat als `unsafe=true` en niet gemarkeerd als behandeld, dat run niet als "done" gemarkeerd kan worden.
27. Zorg dat `stop.log` daadwerkelijk geschreven wordt op die momenten en test de gedrag (bv. simuleer een recipe met een ongeadresseerde gevaarlijke claim, kijk dat het systeem stopt en niet doorgaat).
28. Breid de logging-tags uit: laten we de agent daadwerkelijk `[AMBIGUOUS]` en anderen in hun JSON/log opnemen en verifieer dat de mens ze ziet.

29. **Outcome:** Een robuustere pipeline die niet alleen functioneel werkt, maar ook **veilig faalt** als iets niet klopt. We minimaliseren vals positieven via calibratie (we willen niet om elke pietluttigheid stoppen). Dit is afregelwerk.

30. **User interface & workflow tools (Week 12-16):** Verbeter het gebruikersgemak voor de redactie:

31. Introduceer een eenvoudige CLI-menu of TUI (Text-based UI) zodat redacteuren zonder diep technische kennis een run kunnen starten of de status kunnen checken. Bijvoorbeeld een script dat vraagt "Welk recept wil je verwerken?" en stap voor stap de output toont.
32. Bouw het genoemde rapport of dashboard voor issue-overzicht. Misschien een small Flask-app of gewoon markdown export + open in browser.
33. Implementeer de `review_report.py` die uit sandbox JSON een leesbare lijst maakt.

34. **Outcome:** De redactie kan nu prettiger met de resultaten omgaan. Dit verhoogt acceptatie en zorgt dat de focus op inhoud blijft, niet op het ontcijferen van JSON.

35. **Integratie Phase 2 voorbereiden (Week 16-20):** Phase 2 (publieksedite) zal toestaan dat bepaalde recepten herschreven/verbeterd worden voor leesbaarheid, mits traceable [95]. Om hierop voorbereid te zijn:

36. Evalueer welke delen van de pipeline hergebruikt kunnen worden en welke nieuwe nodig zijn (bijv. een *Rewrite Suggestion Agent* die optioneel alternatieve formuleringen geeft, maar dat is riskant gebied).
37. Begin met documenteren hoe een "moderniseringsbeslissing" zou worden genomen in Phase 2: waarschijnlijk via hetzelfde signaal-gate-closure principe, maar nu met toestemming tot grotere wijzigingen indien gemotiveerd.
38. Maak dit onderdeel van de roadmap, maar voer nog niet daadwerkelijk grootschalig uit – het is meer planning. Misschien een pilot op 1 recept: laat de readability agent een voorgestelde herschreven versie doen in sandbox, en laat redactie beslissen.

39. **Outcome:** Een plan voor Phase 2 zodat investeringen nu (bijv. logging, gating) daarmee compatibel zijn.

40. **Formaliseren Decision Records (Phase-7 style) (Week 20-24):** Naarmate er veel besluiten zijn genomen (glossary keuzes, controverses opgelost), is het zinnig een begin te maken met formele beslissingdocumenten:

41. Creeër een template `docs/decisions/YYYY/<topic>_decision.md` waar men gestructureerd een besluit kan vastleggen (bv. id, type, excerpt, artefacts, rationale – zie voorbeeld in CANONICAL_INDEX.md [35]).
42. Herschrijf enkele al voltooide cases in dit format als oefening. Koppel ze via de index als test.
43. Pas de index in CANONICAL_INDEX.md aan om bijvoorbeeld de eerste 2026 beslissingen op te nemen.

44. **Outcome:** Het begin van een “beslissingsregister” is gemaakt. Dit is vooral documentair, maar het laat zien dat ons systeem uiteindelijk in staat is van alle proposals naar een geconsolideerde kennisbank te groeien.

45. **Testing, QA en Performance tuning (Week 20-24, parallel):**

46. Voer uitgebreide tests uit: unit tests voor validators, sample runs voor verschillende hoofdstukken, controleer dat alle STOP triggers werken.
47. Optimaliseer waar nodig: bijv. caching van model outputs voor identieke inputs (mocht dat voorkomen), of parallel draaien van twee agents als ze duidelijk onafhankelijk zijn (CrewAI kan parallel tasks als dat zinvol is).
48. Zorg ook voor backup-strategie: als Ollama of een model faalt, hebben we een fallback (misschien een kleinere model of een retry).

49. **Outcome:** Systeem is stabiel en klaar voor intensief gebruik op alle content.

50. **Volledige rollout op alle content en iteratie (Week 24+):**

    - Nu alles klaar is, doorloopt het team daadwerkelijk alle recepten door de pipeline. Dit zal iteratief gaan: run een batch, review en fix, lessons learned -> bijstellen prompts of regels, volgende batch.
    - Regelmatige evaluaties: werkt dit echt kostenverlagend en kwaliteitsverhogend? Pas desnoods de scope van sommige agents aan (als een agent te veel false positives geeft, misschien beter uitschakelen en overlaten aan mens tot we het verbeterd hebben).
    - Documenteer gaandeweg best practices in `docs/WORKFLOW.md` of een handleiding voor nieuwe redacteuren.

Deze roadmap zorgt voor een geleidelijke implementatie waarbij steeds duidelijker wordt wat de AI-ondersteuning brengt en hoe de governancekaders in de praktijk functioneren. Door vroeg een slice werkend te hebben (stap 3) krijgen we feedback van de redacteuren, en door de mechanieken stap voor stap strakker te trekken (stap 4-5) borgen we compliance. Tegelijk blijven we pragmatisch: het einddoel is een mooi vertaald en goed geannoteerd *Mustika Rasa*, maar via deze technische architectuur gebeurt dat op een **controleerbare, reproduceerbare en schaalbare** manier – volledig lokaal op betaalbare hardware, met de mens als kapitein en de AI als waardevolle bemanning.

---

[1] [4] [5] Mustikarasa System Architecture Overview (TOGAF-Based Approach).pdf
file://file_00000000025071f4a6714b3d3c5b6a2e

[2] [6] [7] [8] [12] [13] [17] [18] [23] [24] [29] [55] [67] [68] [70] [94] [95] VISION_AND_STRATEGY.md
file://file_00000000221c71f4bb8b13404c3bcce0

3 16 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 EDITORIAL_CAPABILITIES.md
file://file_00000000d66871f4ba60d769e6d71a12

9 10 11 GOV_A_B_CLASSIFICATION.md
file://file_0000000071a4720a92e76461da641946

14 15 28 41 42 43 44 45 46 47 48 49 71 72 AGENT_PROMPT_PATTERN_P5.md
file://file_00000000d1fc71f4815ac4ff2eb93af8

19 20 30 31 50 51 53 54 56 SIGNAL_GATE_CLOSURE.md
file://file_000000000e5471f4ad8fc5b2a81c841a

21 22 32 33 36 37 38 39 63 65 66 AGENT_RUNTIME_SAFETY_BRIDGE.md
file://file_00000000300071f48f60ff0b55b9df42

25 AGENTS_AS_SENSORS.md
file://file_00000000a16471f4b0d19ea7d67f559c

26 27 59 60 61 62 AGENT_AUTONOMY_ENVELOPE.md
file://file_000000004c9c71f4bdd590cc7eec00d3

34 35 40 69 CANONICAL_INDEX.md
file://file_00000000a71c71f48758393ec5e1f4d0

52 92 93 Installation - CrewAI
https://docs.crewai.com/en/installation

57 58 SAFETY_M4_STOP_CONDITIONS.md
file://file_00000000f21071f492a00985cd9e9913

64 SAFETY_M4_GATE_AND_CLOSURE_RULES.md
file://file_00000000e3dc71f48bf52f9ce548f59b