Template Madlib: Deterministic Token Injection for Conditional IMRAD Manuscripts

A public exemplar for source-owned lexical composition

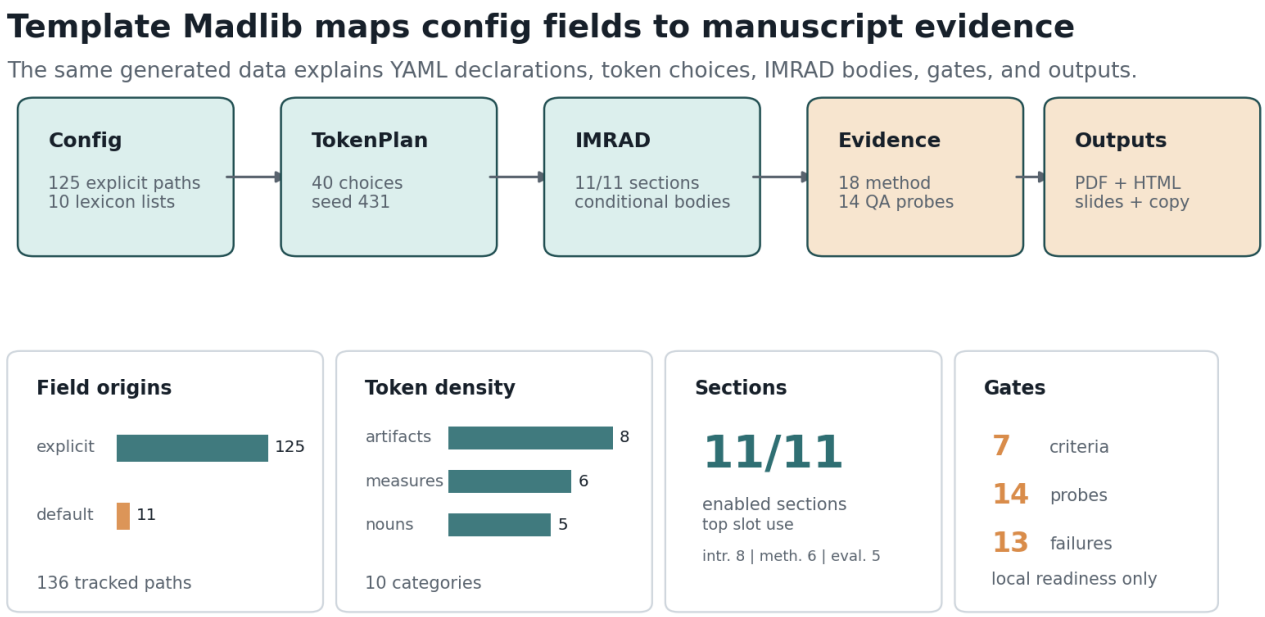
Daniel Ari Friedman

Active Inference Institute

daniel@activeinference.institute

ORCID: 0000-0001-6232-9096

2026-06-21



Cover claim

The figure is generated from the same source-owned inventory, token plan, and QA schema that hydrate the manuscript.

No external validation, DOI, or reader-quality result is implied by this local render.

Contents

1 Abstract

This exemplar asks whether a reviewable pipeline can hydrate a complete IMRAD manuscript from configuration-owned lexical data while preserving an audit trail that remains readable before and after rendering. The project deliberately keeps playful Mad Lib mechanics inside a serious reproducibility contract: the manuscript shell names large placeholders, the config declares allowable language, and the source code decides what text is emitted.

The committed seed is `431` and the current schema expands 22 slot rule(s) into 40 token choice(s) across 10 lexicon categories. The configured narrative moves are state the manuscript-generation problem, name the deterministic intervention, and summarize the audit surface. The central hypothesis is: Deterministic lexical injection can generate a complete conditional IMRAD manuscript while preserving token provenance, section intent, and audit-ready method evidence.

The result is not a claim that lexical substitution creates scholarship. It is a worked template for conditional manuscript assembly: section enablement, token provenance, figure registration, and unresolved-placeholder checks all become inspectable artifacts before the shared renderer produces PDF, HTML, and slides.

2 Introduction: Lexicon as Data and Manuscript as Build Artifact

Mad Lib style generation is usually treated as a toy because it foregrounds the visible blank rather than the source of the replacement. In a research pipeline, the blank is not the hard part. The hard part is making every replacement reviewable, deterministic, and honest about what it can support. `template_madlib` turns that constraint into the subject of the exemplar.

The project treats nouns such as pipeline, protocol, section, lexicon and verbs such as hydrate, condition, bind, bind as versioned data. Changing a lexicon entry is therefore closer to changing an input table than editing prose in place. That distinction matters because the generated manuscript can be rerendered, diffed, and validated without asking the reader to trust an invisible drafting session.

The introduction is configured to separate playful Mad Lib syntax from research claims, identify drift between prose and source data, frame configuration as an inspectable dataset, and position conditional prose as a reproducibility problem. Those moves keep the manuscript from pretending to be an open-ended language model. It is a bounded template: authors declare categories, slots, section switches, method steps, and claim boundaries; source code transforms those declarations into manuscript bodies and evidence tables.

This exemplar is useful for protocols, educational scaffolds, review forms, templated reports, and other documents where conditional text is unavoidable but should never become untraceable. The same pattern can be extended with domain-specific validators while leaving the shared rendering infrastructure untouched.

2.1 Contribution Ledger

Claim	Boundary
A Mad Lib manuscript can remain reproducible when the lexicon is treated as data.	Local exemplar claim; no live DOI or standalone release implied.
Conditional IMRAD section bodies can be rendered without shared renderer changes.	Local exemplar claim; no live DOI or standalone release implied.
Large-grain manuscript variables can preserve author-readable source files while still producing a complete manuscript.	Local exemplar claim; no live DOI or standalone release implied.
Token provenance can connect playful lexical substitution to serious publication hygiene.	Local exemplar claim; no live DOI or standalone release implied.
A generated Methods section can be methodologically useful when protocol rows, phases, figures, and validation gates share one config-owned source.	Local exemplar claim; no live DOI or standalone release implied.
Configured-field origin tracking can make loader defaults visible enough for reviewers and downstream forks.	Local exemplar claim; no live DOI or standalone release implied.
Pipeline-owned output regeneration can keep PDF, HTML, slides, data, reports, and copied deliverables aligned without hand-editing generated artifacts.	Local exemplar claim; no live DOI or standalone release implied.
A generated-method exemplar can make review handoff auditable when the review packet includes source config, data artifacts, figures, validation results, and copy statistics.	Local exemplar claim; no live DOI or standalone release implied.
Fork migration guidance can reduce overclaiming when it names the source, test, validator, and evidence surfaces that must change before domain use.	Local exemplar claim; no live DOI or standalone release implied.

3 Methods: Source-Owned Token Injection and Conditional IMRAD Assembly

The method is conditional section hydration. Each slot combines the seed, slot name, category, ordinal, and full category list into a SHA-256 digest; the digest indexes the configured category. Including the full category list in the digest input means that a lexicon edit can change the plan in a deterministic and reviewable way instead of silently preserving stale output.

The deterministic digest recipe is deliberately narrow. It does not sample from ambient prose, project history, or renderer state; it uses only the committed seed, the slot declaration, the category name, the ordinal for repeated slots, and the ordered category inventory. A fork can therefore explain a changed token choice as a changed seed, slot, or lexicon row rather than as an opaque generation event.

The first review scenario is declared before generation. The project names its scope as a local exemplar, records enabled sections, keeps DOI and publication claims blank, and treats the copy stage as a review handoff. That ordering matters because a reader should know the allowed claim boundary before inspecting fluent generated text.

The governing constraint is publication claims stay local until release. The source manuscript is intentionally sparse: it contains section titles and large-grain placeholders, not generated claims. The project script first validates the `madlib:` block, expands slots into a `TokenPlan`, builds section bodies, writes artifact JSON, emits a figure registry, and only then writes hydrated Markdown under `output/manuscript/`.

The configured method moves are validate config before composition, declare review scenario and method invariants, separate explicit YAML fields from loader defaults, govern lexicon categories as reviewable data, expand slots through seeded digest selection, allocate slot choices to enabled manuscript sections, compose conditional section bodies, assemble method evidence tables and visual audit figures, align generated claims with the claim ledger, emit evidence artifacts before rendering, verify that no unresolved placeholders remain, assemble a reviewer packet from regenerated artifacts, and preserve human review before publication claims. The protocol sequence is Ingest declared manuscript schema produces `MadlibConfig`; Declare review scenario produces `review scenario`; Track field origin produces `explicit/default path inventory`; Govern lexicon categories produces `validated lexicon`; Construct digest selection material produces `digest input records`; Record selection invariants produces `selection invariant set`; Expand slot declarations produces `TokenPlan`; Apply section conditions produces `enabled section set`; Compose conditional IMRAD bodies produces `section variables`; Assemble evidence tables produces `Markdown evidence tables`; Align claims with evidence ledger produces `claim-aligned evidence surface`; Generate visual audit surface produces `registered figure set`; Emit machine-readable artifacts produces `output/data, output/reports, and output/figures`; Hydrate manuscript shell produces `output/manuscript`; Render and validate deliverables produces `validated project output`; Assemble reviewer packet produces `review packet`; Copy review surface produces `copied publication-review bundle`; Document fork migration produces `fork migration notes`. These steps make the method auditable from three directions: tests inspect the Python behavior, generated artifacts expose the token plan, and manuscript validation confirms that no unresolved placeholder survives into rendered outputs.

The config-origin inventory currently separates 125 explicit YAML path(s) from 11 loader-defaulted path(s). Treating origin as method evidence prevents a rendered field from looking equally authored when it was actually inherited from the loader. The same inventory drives configured-field tables and figures, so reviewers can inspect which schema blocks were intentionally set before judging the generated prose.

Method invariants are reviewed as their own artifact. Token choices are allowed to change when the seed, slot name, category, ordinal, or ordered category inventory changes; they are not allowed to change because PDF rendering, HTML rendering, file-copy order, or hand-edited output changed. This separates generation logic from presentation logic.

Lexicon governance is handled as data governance. Required categories must be nonempty, optional categories remain project-owned when declared, and the selected 40 token choice(s) stay bound to 10 configured category list(s). The slot-to-section allocation is abstract: 3, authoring_contract: 2, configuration: 1, discussion: 3, evaluation: 5, introduction: 8, limitations: 3, methods: 6, reproducibility: 2, results: 5, scope: 2, which lets the Methods, Results, and provenance tables state where each lexical decision enters the manuscript.

Conditional section generation is handled before prose assembly. A disabled section does not vanish and does not borrow claims from an enabled section; it resolves to an explicit statement that names the controlling `madlib.section_conditions` key. That behavior keeps negative or excluded material visible to reviewers.

Evidence tables and visual audit figures are generated from the same config and `TokenPlan`. Enabled visualizations are configured_field_matrix, section_configuration_heatmap, field_origin_summary, token_injection_flow, section_token_allocation, provenance_trace_map, quality_gate_matrix; they summarize field origin, token density, injection flow, section allocation, provenance, and quality gates without adding independent claims. Figure registration is therefore a method step: every manuscript image has to be written, registered, and validated as part of the reproducible render path.

Claim-ledger alignment is part of composition. The contribution table can make local method claims, but publication, empirical, reader-quality, and domain-specific claims must either point to evidence or remain explicit non-claims. This is why the claim ledger, audit rules, limitations, and authoring contract are generated beside the Methods rather than written after the fact.

Evaluation is part of the method rather than an afterthought. The config declares quality probes (Method row completeness, Field-origin visibility, Placeholder survival, Provenance completeness, Section-switch observability, Figure registry coverage, Method-figure alignment, Evidence cleanliness, Fork readiness, Copied-output parity, Digest invariant review, Claim-ledger alignment, Review

packet completeness, Fork migration sufficiency) and failure modes (Unresolved placeholder, Overclaimed generated prose, Config-source drift, Figure provenance gap, Domain misuse, Method row drift, Field-origin opacity, Visual-method mismatch, Fork without validators, Digest invariant drift, Claim ledger omission, Review packet incompleteness, Fork migration ambiguity); the source turns them into tables and validation checks the rendered surface. That means methods, results, evaluation, and limitations all share one source-owned schema instead of drifting as independent prose.

The method is organized around design principles: Configuration owns prose choices, Method surface is config-owned, Token choice is deterministic, Field origin is evidence, Sections are conditional but visible, Visual audit follows data, Generated output is disposable, Claim boundaries travel with prose, Forks must add validators, Invariants precede rendering, Diffs are review objects, Review packet is a method artifact, Fork migration is part of the method. These principles prevent the Mad Lib surface from becoming a hidden authoring channel. They require the visible manuscript to stay downstream of declared inputs, the generated outputs to remain disposable, and the audit surface to be broad enough for a reviewer to reconstruct how a sentence reached the PDF.

The operational phases are Schema intake maps `manuscript/config.yaml` to `MadlibConfig`; Scenario declaration maps `MadlibConfig` to `review scenario`; Field-origin inventory maps `MadlibConfig and raw YAML keys` to `configured_field_inventory.json`; Lexicon validation maps `madlib.lexicon and madlib.slots` to `validated slot inventory`; Digest token planning maps `MadlibConfig` to `TokenPlan`; Invariant review maps `TokenPlan and method protocol` to `selection invariant set`; Slot-to-section allocation maps `TokenPlan` to `section token counts`; Section composition maps `TokenPlan and narrative moves` to `manuscript variable map`; Evidence table assembly maps `MadlibConfig and TokenPlan` to `manuscript_variables.json`; Claim-ledger alignment maps `MadlibConfig, generated prose, and data/claim_ledger.yaml` to `claim-aligned evidence surface`; Visualization emission maps `MadlibConfig, TokenPlan, and configured-field inventory` to `output/figures and figure_registry.json`; Artifact emission maps `MadlibConfig and TokenPlan` to `output/data, output/reports, and output/figures`; Manuscript hydration maps `source manuscript shells and manuscript_variables.json` to `hydrated Markdown manuscript`; Render maps `output/manuscript` to `output/pdf, output/web, and output/slides`; Validate and copy maps `project output directories` to `output/templates/template_madlib`; Review packet assembly maps `validated project output and copy statistics` to `review packet`; Fork contract documentation maps `source docs, authoring contract, and claim ledger` to `fork migration notes`. Each phase has an explicit input, transformation, output, and guard. This makes the pipeline explainable at manuscript scale: a reader can follow the path from YAML declarations to token choices, from token choices to section bodies, from section bodies to rendered artifacts, and from rendered artifacts to validation reports.

The reviewer packet is also a method artifact. The handoff surface is hydrated Markdown, combined PDF, web output, slides, figures, data JSON, reports, validation results, and copy statistics; a PDF alone is insufficient because it cannot show the token inventory, field-origin inventory, figure registry, validation report, or copied-output statistics. The declared authoring obligations are Review generated claims, Review config diffs, Extend claim evidence, Add domain validators, Rerun the full project path, Review method invariants, Assemble reviewer packet, Write fork migration notes, which convert that packet into review work a human can actually perform.

The claim-boundary contract is also generated. The audit-rule list contains 12 rule(s), and the contribution table binds each local claim to a non-publication boundary. The final copy stage is a human-review handoff, not proof that the Mad Lib surface is empirically valid or ready for a standalone release.

Fork migration closes the method. A downstream project should update config rows, source-owned composition, validators, claim-ledger entries, and documentation before replacing exemplar vocabulary with domain claims. Without that migration work, the fork has only changed words, not the evidential status of the generated manuscript.

3.1 Design Principles

Principle	Rationale	Manuscript effect
Configuration owns prose choices	Reviewers can inspect the declared language surface before generation.	Large-grain manuscript variables are generated from YAML and project source.
Method surface is config-owned	A fork should change method protocol rows before changing generated Methods prose.	The Methods tables and body summarize method_protocol and pipeline_phases from YAML.
Token choice is deterministic	A fixed seed and lexicon must produce the same injection plan across reruns.	Token inventory rows can be regenerated and diffed.
Field origin is evidence	A generated manuscript should distinguish authored YAML fields from loader defaults.	Configured-field tables and figures report explicit and defaulted paths.
Sections are conditional but visible	Disabled material should be auditable rather than silently absent.	Every disabled section resolves to an explicit disabled-section body.
Visual audit follows data	Figures should explain the generated method without becoming decorative claims.	Cover, flow, allocation, provenance, gate, and field-origin figures are regenerated from artifacts.
Generated output is disposable	The durable artifact is the regeneration contract, not hand-edited output files.	Output Markdown, PDF, HTML, and slides are rebuilt from source inputs.

Principle	Rationale	Manuscript effect
Claim boundaries travel with prose	Generated text can otherwise imply validation that no artifact supports.	Contribution, limitation, failure-mode, and authoring tables carry boundary text.
Forks must add validators	Domain claims need domain evidence beyond this exemplar's generic regeneration gates.	Scope, limitations, and authoring contract require validators before domain claims.
Invariants precede rendering	Readers need to know which inputs are allowed to alter generated tokens before they inspect output.	Methods names the digest inputs and excludes renderer state from token choice.
Diffs are review objects	Config, token inventory, manuscript variables, figures, and copied outputs should be diffable review surfaces.	Methods and reproducibility text describe review packet assembly after regeneration.
Review packet is a method artifact	A PDF alone is insufficient evidence for a generated-method exemplar.	Authoring contract and validation sections treat data, reports, figures, and logs as part of review.
Fork migration is part of the method	A public exemplar should teach authors what must change before domain use.	Standalone notes, authoring obligations, and claim-ledger boundaries name fork responsibilities.

3.2 Operational Phases

Phase	Input	Transformation	Output	Guard
Schema intake	`manuscript/config.yaml`	Load paper metadata and validate the madlib schema before generation.	`MadlibConfig`	config parser tests
Scenario declaration	`MadlibConfig`	Summarize local scope, enabled sections, claim boundaries, and review handoff expectations.	`review scenario`	method protocol and contribution table tests
Field-origin inventory	`MadlibConfig and raw YAML keys`	Classify supported paths as explicit or defaulted.	`configured_field_inventory.json`	configured-field inventory tests
Lexicon validation	`madlib.lexicon and madlib.slots`	Reject empty required categories and slot references to missing categories.	`validated slot inventory`	malformed-config tests
Digest token planning	`MadlibConfig`	Hash seed, slot name, category, ordinal, and category inventory.	`TokenPlan`	seed-stability tests
Invariant review	`TokenPlan and method protocol`	Confirm allowed token-choice inputs are documented and isolated from renderer state.	`selection invariant set`	token determinism and Methods prose tests
Slot-to-section allocation	`TokenPlan`	Assign each selected token to its configured manuscript section.	`section token counts`	provenance and allocation tests
Section composition	`TokenPlan and narrative moves`	Build conditional section bodies, titles, and evidence tables.	`manuscript variable map`	placeholder-coverage tests
Evidence table assembly	`MadlibConfig and TokenPlan`	Render protocol, phase, audit, token, section, provenance, and configured-field tables.	`manuscript_variables.json`	composition table tests
Claim-ledger alignment	`MadlibConfig, generated prose, and data/claim_ledger.yaml`	Check local claims and non-claims against source-owned evidence rows.	`claim-aligned evidence surface`	claim ledger and evidence registry review

Phase	Input	Transformation	Output	Guard
Visualization emission	`MadlibConfig, TokenPlan, and configured-field inventory`	Generate cover, flow, allocation, provenance, gate, and configured-field figures.	`output/figures and figure_registry.json`	nonblank figure tests
Artifact emission	`MadlibConfig and TokenPlan`	Write inventory, section plan, injection trace, summary, cover overview, manuscript figures, and registry.	`output/data, output/reports, and output/figures`	artifact-writing tests
Manuscript hydration	`source manuscript shells and manuscript_variables.json`	Resolve large-grain placeholders into output/manuscript.	`hydrated Markdown manuscript`	unresolved-token scan
Render	`output/manuscript`	Render PDF, HTML, and slides through the shared template pipeline.	`output/pdf, output/web, and output/slides`	render command
Validate and copy	`project output directories`	Validate files, registries, evidence, overlays, and copied deliverables.	`output/templates/template_madlib`	validation and copy commands
Review packet assembly	`validated project output and copy statistics`	Group manuscript, web, slides, figures, data, reports, validation results, and copy statistics as the review surface.	`review packet`	copied-output validation
Fork contract documentation	`source docs, authoring contract, and claim ledger`	State which config, source, test, validator, and evidence surfaces a fork must change before domain claims.	`fork migration notes`	documentation and claim-ledger tests

3.3 Protocol Steps

Step	Action	Evidence	Output
Ingest declared manuscript schema	Parse paper metadata and the madlib block from manuscript/config.yaml before any prose or figures are composed.	Config validation tests and MadlibConfig construction from the committed YAML.	`MadlibConfig`
Declare review scenario	Name the manuscript scope, local claim boundary, enabled sections, and intended reviewer handoff before token generation.	section_plan.json, contribution table, and authoring contract rows.	`review scenario`
Track field origin	Record every supported madlib path as explicit when it appears in YAML or defaulted when the loader supplies it.	configured_field_inventory.json and configured-field origin tests.	`explicit/default path inventory`
Govern lexicon categories	Reject empty required categories, preserve project-owned optional categories, and treat every lexical list as source data.	Malformed-config tests and lexicon rows in token_inventory.json.	`validated lexicon`
Construct digest selection material	Combine seed, slot name, category, ordinal, and the full category inventory into a deterministic digest input.	Seed-stability and category-sensitivity tests.	`digest input records`

Step	Action	Evidence	Output
Record selection invariants	State the invariant inputs that are allowed to change token choices and the renderer state that is not allowed to affect them.	Token determinism tests and Methods digest prose.	`selection invariant set`
Expand slot declarations	Resolve every slot count into one or more token choices and assign each selected value to its configured manuscript section.	TokenPlan construction, provenance trace, and section allocation figure.	`TokenPlan`
Apply section conditions	Evaluate section switches before prose assembly so disabled sections receive explicit disabled-section bodies.	Section-condition tests and output/data/section_plan.json.	`enabled section set`
Compose conditional IMRAD bodies	Build large-grain section variables from narrative moves, selected tokens, section switches, and local claim boundaries.	Generated manuscript variables and hydrated output/manuscript Markdown.	`section variables`
Assemble evidence tables	Render method protocol, pipeline phase, design principle, audit rule, token inventory, section plan, and provenance tables from config and TokenPlan.	Composition tests and generated manuscript_variables.json table entries.	`Markdown evidence tables`
Align claims with evidence ledger	Keep generated method, visualization, and publication-boundary claims tied to config, source modules, generated artifacts, or explicit non-claim boundaries.	data/claim_ledger.yaml and evidence registry validation.	`claim-aligned evidence surface`
Generate visual audit surface	Write the cover overview, token-injection flow, section allocation, provenance trace, quality gate, and configured-field figures from generated data.	Nonblank figure tests and ../figures/figure_registry.json.	`registered figure set`
Emit machine-readable artifacts	Write token inventory, section plan, configured-field inventory, injection trace, summary reports, validation inputs, and figure registry.	Artifact-writing tests, artifact manifest, and validation reports.	`output/data, output/reports, and output/figures`
Hydrate manuscript shell	Write manuscript_variables.json and resolve the source Markdown shells into output/manuscript.	Unresolved-token scan and render validation.	`output/manuscript`
Render and validate deliverables	Render PDF, HTML, and slides through shared infrastructure, then validate PDFs, Markdown, figure registry, evidence registry, design overlays, and artifact manifest.	Stage 03 render log and Stage 04 validation report.	`validated project output`
Assemble reviewer packet	Treat hydrated Markdown, rendered PDF, web output, slides, figures, data, reports, validation logs, and copied output statistics as one review packet.	Stage 04 validation report and Stage 05 output_statistics.json.	`review packet`

Step	Action	Evidence	Output
Copy review surface	Copy validated deliverables into output/templates/template_madlib only after validation passes.	Stage 05 copy statistics and copied-output validation.	`copied publication-review bundle`
Document fork migration	Record what downstream authors must change when moving from exemplar token injection to a domain-specific report.	README, STANDALONE notes, manuscript README, Authoring Contract, and claim ledger boundary rows.	`fork migration notes`

3.4 Section Plan

Section	Render title	Enabled	Token choices	Narrative moves
abstract	Abstract	True	3	state the manuscript-generation problem, name the deterministic intervention, summarize the audit surface
introduction	Introduction: Lexicon as Data and Manuscript as Build Artifact	True	8	separate playful Mad Lib syntax from research claims, identify drift between prose and source data, frame configuration as an inspectable dataset, position conditional prose as a reproducibility problem

Section	Render title	Enabled	Token choices	Narrative moves
methods	Methods: Source-Owned Token Injection and Conditional IMRAD Assembly	True	6	validate config before composition, declare review scenario and method invariants, separate explicit YAML fields from loader defaults, govern lexicon categories as reviewable data, expand slots through seeded digest selection, allocate slot choices to enabled manuscript sections, compose conditional section bodies, assemble method evidence tables and visual audit figures, align generated claims with the claim ledger, emit evidence artifacts before rendering, verify that no unresolved placeholders remain, assemble a reviewer packet from regenerated artifacts, preserve human review before publication claims
results	Results: Provenance, Density, and Resolved Manuscript Surface	True	5	report token density, show resolved section coverage, bind every manuscript token to provenance, connect the figure and inventory to the same plan
discussion	Discussion: Accountability Boundaries for Generated Prose	True	3	bound the scholarly claim, describe useful adaptation cases, name misuse modes, preserve human authorship responsibility
configuration	Configuration: Schema-Controlled Lexicon, Slots, and Narrative Moves	True	1	document schema ownership, show title and switch behavior, record generated counts from code
evaluation	Evaluation: Gate Criteria, QA Probes, and Failure Discovery	True	5	name readiness criteria, connect criteria to artifacts, separate local checks from publication readiness, make failure probes visible

Section	Render title	Enabled	Token choices	Narrative moves
reproducibility	Reproducibility: Seeded Regeneration and Artifact Trace	True	2	fix seed and config hash, write machine-readable artifacts, rerender through the shared pipeline, copy outputs only after validation
limitations	Limitations: Non-Claims, Misuse Modes, and Human Review	True	3	state non-claims, identify misuse modes, preserve human review, require domain validators for domain claims
scope	Scope: Related Generators and Responsible Forking	True	2	distinguish generation from truth, limit publication claims, point to local evidence, explain responsible forking
authoring_contract	Authoring Contract: Human Review and Forking Obligations	True	2	state human responsibilities, name fork obligations, connect review to generated evidence, require domain validators before domain claims, document fork migration notes

3.5 Audit Rules

Rule	Enforcement surface
R1	Every manuscript placeholder must be generated by source code.
R2	Every generated token choice must carry category and config-key provenance.
R3	Every method protocol row must identify an action, evidence artifact, and output.
R4	Every pipeline phase must identify an input, transformation, output, and guard.
R5	Every visible configured field must be classified as explicit or defaulted.
R6	Every disabled section must resolve to an explicit disabled-section body.
R7	Every figure reference must be backed by a generated figure registry entry.
R8	Every fork that adds domain claims must add domain validators and claim-ledger evidence.
R9	Every publication claim must stay local unless a live DOI or release exists.
R10	Every token-selection explanation must name only seed, slot, category, ordinal, and ordered category inventory as digest inputs.
R11	Every review handoff must include generated data, reports, figures, validation results, and copy statistics alongside PDF or HTML.

Rule	Enforcement surface
R12	Every fork migration note must name config, source, test, validator, pipeline, and claim-ledger obligations.

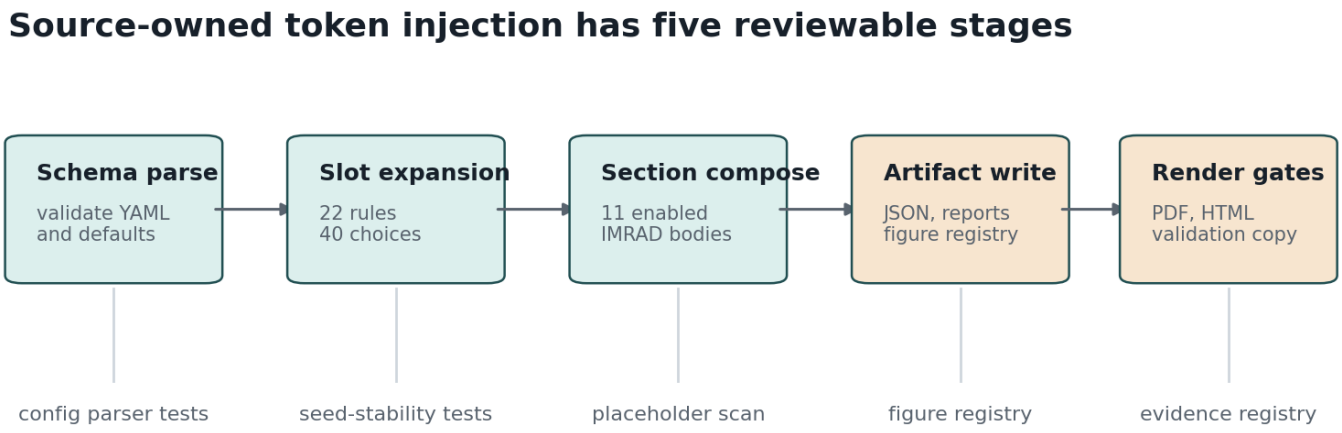


Figure 1: Deterministic token-injection flow

4 Results: Provenance, Density, and Resolved Manuscript Surface

The generated plan enables 11 of 11 manuscript sections and fills 40 token choice(s). Category density is adjectives: 2, artifacts: 8, audiences: 4, constraints: 3, failures: 3, measures: 6, methods: 1, nouns: 5, qualities: 3, verbs: 5. The result figures are generated from the same token plan that writes the inventory table, so visual and tabular claims share one source.

The configured results moves are report token density, show resolved section coverage, bind every manuscript token to provenance, and connect the figure and inventory to the same plan. The important result is therefore not a surprising word choice; it is the survival of traceability through a complete render path. Each token row records the variable, category, selected value, section, and config pointer that produced it.

The resolved manuscript also demonstrates the intended failure boundary. If a manuscript placeholder is added without a corresponding variable, the project test suite detects it. If a figure is referenced without registry support, the output validator reports it. If a generated number lacks evidence support, the evidence registry gate reports it before the copied output stage packages deliverables.

Visualization is enabled for configured_field_matrix, section_configuration_heatmap, field_origin_summary, token_injection_flow, section_token_allocation, provenance_trace_map, quality_gate_matrix. The configured-field figures are generated from the same explicit/default path inventory written to JSON; the pipeline, allocation, provenance, and gate figures are generated from the same token plan and QA schema.

4.1 Token Inventory

Variable	Category	Value	Section	Source
STUDY_ADJECTIVE	adjectives	reviewable	abstract	manuscript/config.yaml#mad
STUDY_NOUN	nouns	pipeline	abstract	manuscript/config.yaml#mad
STUDY_VERB	verbs	hydrate	abstract	manuscript/config.yaml#mad
INTRO_NOUNS_1	nouns	protocol	introduction	manuscript/config.yaml#mad
INTRO_NOUNS_2	nouns	section	introduction	manuscript/config.yaml#mad
INTRO_NOUNS_3	nouns	lexicon	introduction	manuscript/config.yaml#mad
INTRO_NOUNS_4	nouns	artifact	introduction	manuscript/config.yaml#mad
INTRO_VERBS_1	verbs	condition	introduction	manuscript/config.yaml#mad
INTRO_VERBS_2	verbs	bind	introduction	manuscript/config.yaml#mad
INTRO_VERBS_3	verbs	bind	introduction	manuscript/config.yaml#mad
INTRO_VERBS_4	verbs	compose	introduction	manuscript/config.yaml#mad
METHOD_NAME	methods	conditional section hydration	methods	manuscript/config.yaml#mad
METHOD_CONSTRAINT	constraints	publication claims stay local until release	methods	manuscript/config.yaml#mad

Variable	Category	Value	Section	Source
METHOD_ARTIFACT_1	artifacts	token-injection flow	methods	manuscript/config.yaml#mad
METHOD_ARTIFACT_2	artifacts	quality-gate matrix	methods	manuscript/config.yaml#mad
METHOD_QUALITY_1	qualities	claim humility	methods	manuscript/config.yaml#mad
METHOD_QUALITY_2	qualities	render readiness	methods	manuscript/config.yaml#mad
RESULT_MEASURE_1	measures	provenance coverage	results	manuscript/config.yaml#mad
RESULT_MEASURE_2	measures	evidence registry cleanliness	results	manuscript/config.yaml#mad
RESULT_MEASURE_3	measures	category density	results	manuscript/config.yaml#mad
RESULT_ARTIFACT_1	artifacts	configured-field figures	results	manuscript/config.yaml#mad
RESULT_ARTIFACT_2	artifacts	token inventory	results	manuscript/config.yaml#mad
DISCUSSION_ADJECTIVE	adjectives	auditable	discussion	manuscript/config.yaml#mad
DISCUSSION_AUDIENCE_1	audiences	pipeline maintainers	discussion	manuscript/config.yaml#mad
DISCUSSION_AUDIENCE_2	audiences	research educators	discussion	manuscript/config.yaml#mad
CONFIG_CONSTRAINT	constraints	disabled sections retain explicit traceability	configuration	manuscript/config.yaml#mad
EVALUATION_MEASURE_1	measures	copied output readiness	evaluation	manuscript/config.yaml#mad
EVALUATION_MEASURE_2	measures	figure registry completeness	evaluation	manuscript/config.yaml#mad
EVALUATION_MEASURE_3	measures	category density	evaluation	manuscript/config.yaml#mad
EVALUATION_ARTIFACT_1	artifacts	manuscript variable map	evaluation	manuscript/config.yaml#mad
EVALUATION_ARTIFACT_2	artifacts	provenance trace map	evaluation	manuscript/config.yaml#mad
REPRODUCIBILITY_ARTIFACT_1	artifacts	section plan	reproducibility	manuscript/config.yaml#mad
REPRODUCIBILITY_ARTIFACT_2	artifacts	manuscript variable map	reproducibility	manuscript/config.yaml#mad
LIMITATION_FAILURE_1	failures	domain misuse	limitations	manuscript/config.yaml#mad
LIMITATION_FAILURE_2	failures	overclaimed generated prose	limitations	manuscript/config.yaml#mad
LIMITATION_FAILURE_3	failures	figure provenance gap	limitations	manuscript/config.yaml#mad
SCOPE_CONSTRAINT	constraints	all lexicon entries live in config	scope	manuscript/config.yaml#mad
SCOPE_AUDIENCE	audiences	pipeline maintainers	scope	manuscript/config.yaml#mad
AUTHORING_AUDIENCE	audiences	manuscript reviewers	authoring_contract	manuscript/config.yaml#mad
AUTHORING_QUALITY	qualities	render readiness	authoring_contract	manuscript/config.yaml#mad

4.2 Provenance Matrix

Section	Token variables	Source categories
Abstract	STUDY_ADJECTIVE, STUDY_NOUN, STUDY_VERB	adjectives, nouns, verbs
Introduction: Lexicon as Data and Manuscript as Build Artifact	INTRO_NOUNS_1, INTRO_NOUNS_2, INTRO_NOUNS_3, INTRO_NOUNS_4, INTRO_VERBS_1, INTRO_VERBS_2, INTRO_VERBS_3, INTRO_VERBS_4	nouns, verbs
Methods: Source-Owned Token Injection and Conditional IMRAD Assembly	METHOD_NAME, METHOD_CONSTRAINT, METHOD_ARTIFACT_1, METHOD_ARTIFACT_2, METHOD_QUALITY_1, METHOD_QUALITY_2	artifacts, constraints, methods, qualities
Results: Provenance, Density, and Resolved Manuscript Surface	RESULT_MEASURE_1, RESULT_MEASURE_2, RESULT_MEASURE_3, RESULT_ARTIFACT_1, RESULT_ARTIFACT_2	artifacts, measures
Discussion: Accountability Boundaries for Generated Prose	DISCUSSION_ADJECTIVE, DISCUSSION_AUDIENCE_1, DISCUSSION_AUDIENCE_2	adjectives, audiences
Configuration: Schema-Controlled Lexicon, Slots, and Narrative Moves	CONFIG_CONSTRAINT	constraints
Evaluation: Gate Criteria, QA Probes, and Failure Discovery	EVALUATION_MEASURE_1, EVALUATION_MEASURE_2, EVALUATION_MEASURE_3, EVALUATION_ARTIFACT_1, EVALUATION_ARTIFACT_2	artifacts, measures
Reproducibility: Seeded Regeneration and Artifact Trace	REPRODUCIBILITY_ARTIFACT_1, REPRODUCIBILITY_ARTIFACT_2	artifacts
Limitations: Non-Claims, Misuse Modes, and Human Review	LIMITATION_FAILURE_1, LIMITATION_FAILURE_2, LIMITATION_FAILURE_3	failures

Section	Token variables	Source categories
Scope: Related Generators and Responsible Forking	SCOPE_CONSTRAINT, SCOPE_AUDIENCE	audiences, constraints
Authoring Contract: Human Review and Forking Obligations	AUTHORING_AUDIENCE, AUTHORING_QUALITY	audiences, qualities

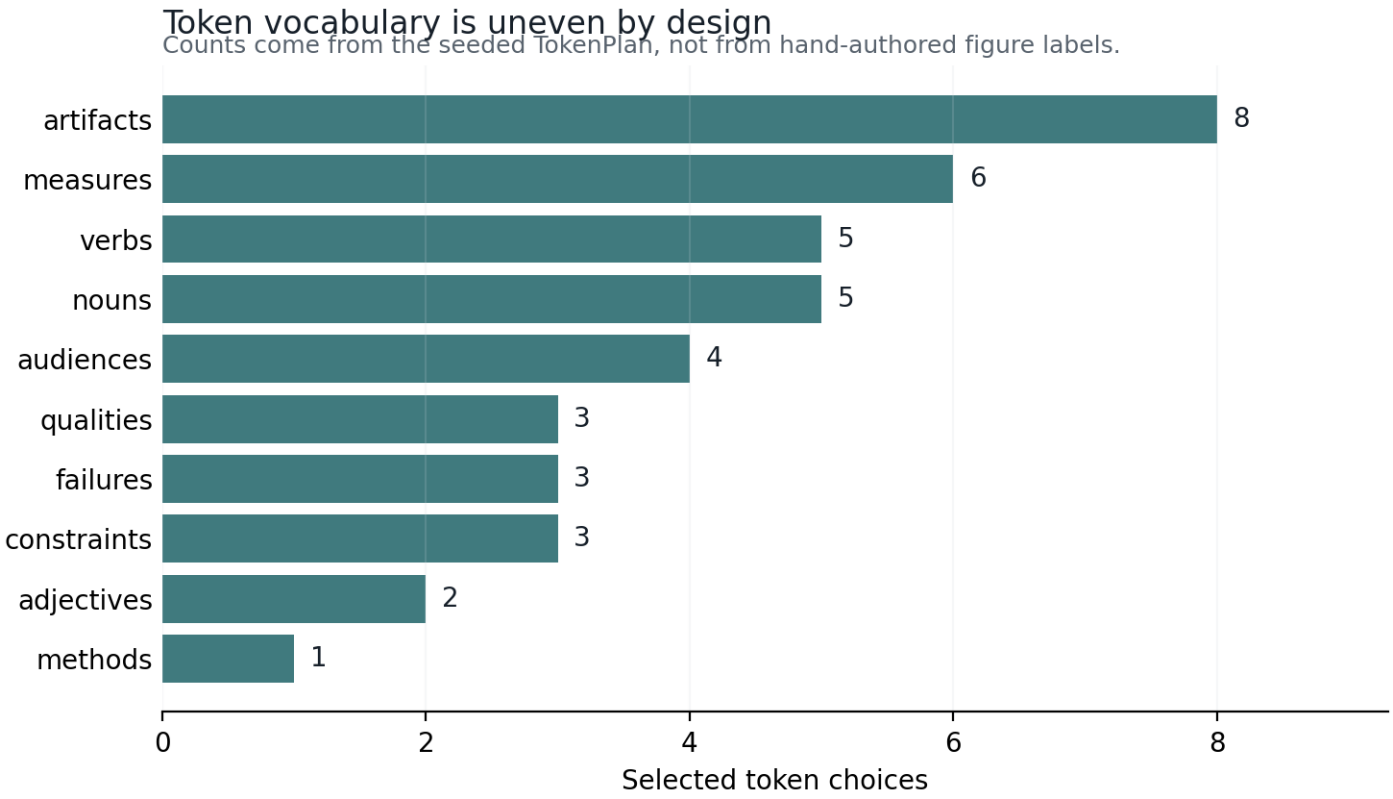


Figure 2: Token category density

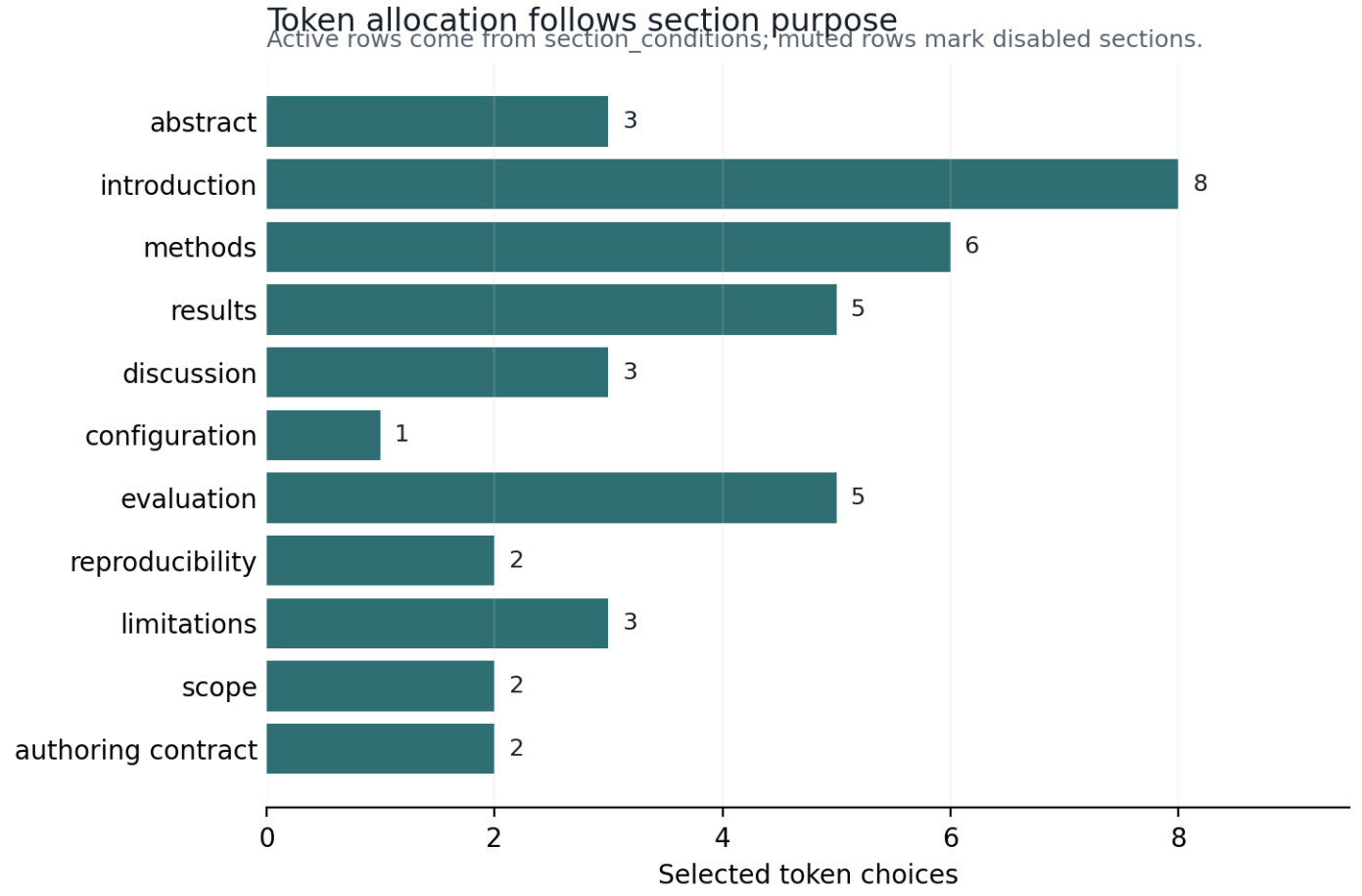


Figure 3: Section token allocation

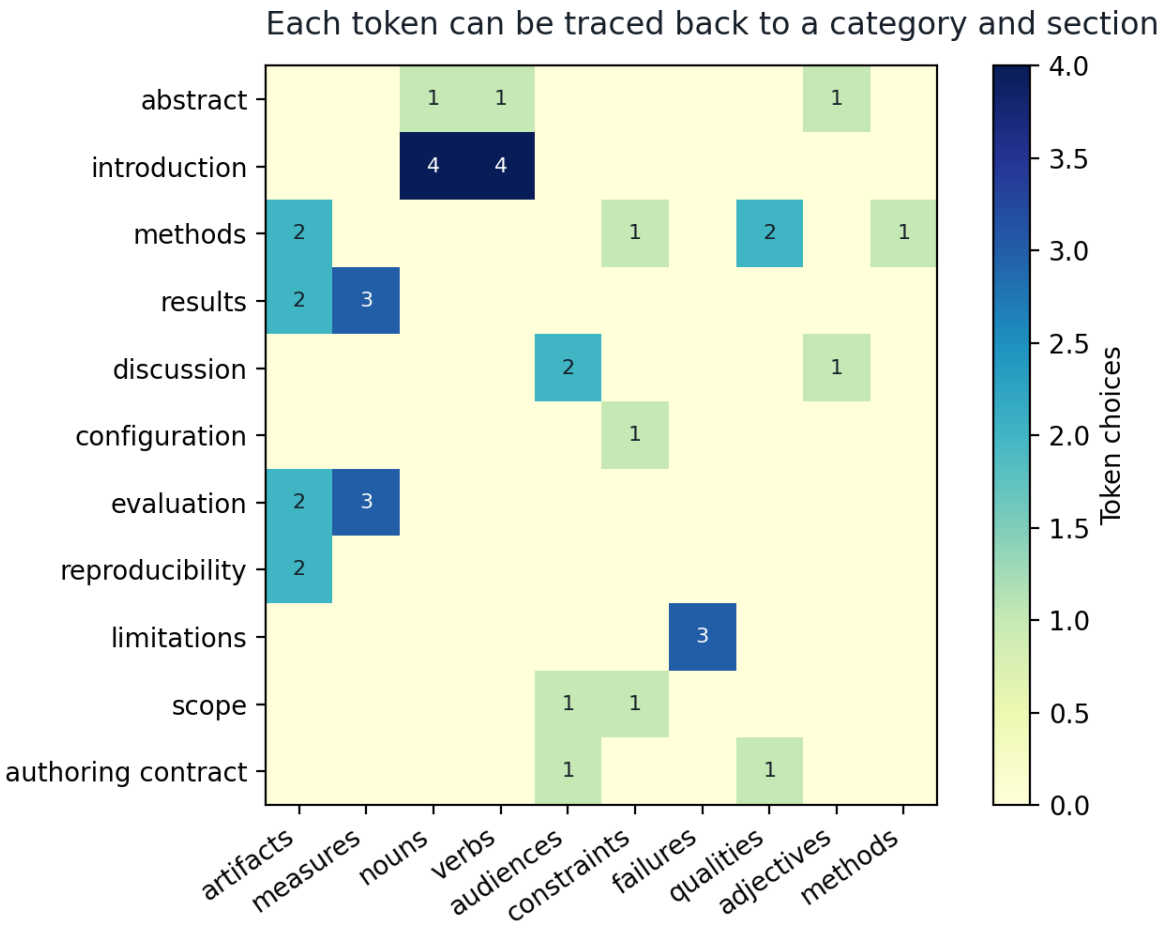


Figure 4: Provenance trace map

5 Discussion: Accountability Boundaries for Generated Prose

The reviewable result is intentionally modest. The exemplar does not claim that random lexical replacement creates scholarship, discovers facts, or substitutes for authorship. It shows that a conditional text generator can be made accountable to configuration, tests, and render-time validation.

The configured discussion moves are bound the scholarly claim, describe useful adaptation cases, name misuse modes, and preserve human authorship responsibility. Useful adaptations include templated empirical reports, structured review memos, classroom exercises, and manuscript sections that need to toggle across protocols. Risky adaptations include hiding weak claims behind fluent generated phrasing or allowing a token category to imply evidence that the project never produced.

The pattern scales only when authors preserve the same ownership boundary. Lexicons should be small enough to review, categories should be named for their manuscript function, and section switches should state what has been removed. When richer language is needed, the next layer should add domain-specific validators rather than relaxing provenance.

6 Configuration: Schema-Controlled Lexicon, Slots, and Narrative Moves

The active composition depth is `deep`. The lexicon exposes 10 category list(s), and the slot declaration expands to 40 concrete token choice(s). Section switches are evaluated before composition so disabled sections cannot silently borrow enabled-section claims.

Configuration owns more than vocabulary. It also owns section titles, narrative moves, method protocol rows, contribution claims, and audit rules. That makes the manuscript shape visible in one YAML file while preserving the rule that source code, not hand-edited output, performs the composition.

The tables in this section expose the declared surface that controls rendering. They are useful during review because a title change, slot expansion, or disabled section appears as a small config diff and a regenerated artifact diff rather than as scattered prose edits.

The configured-field inventory separates 125 explicit YAML path(s) from 11 loader-defaulted path(s). That distinction matters for forks: a field that appears in the rendered manuscript may be intentionally authored in `config.yaml`, or it may be a documented default inherited from the template.

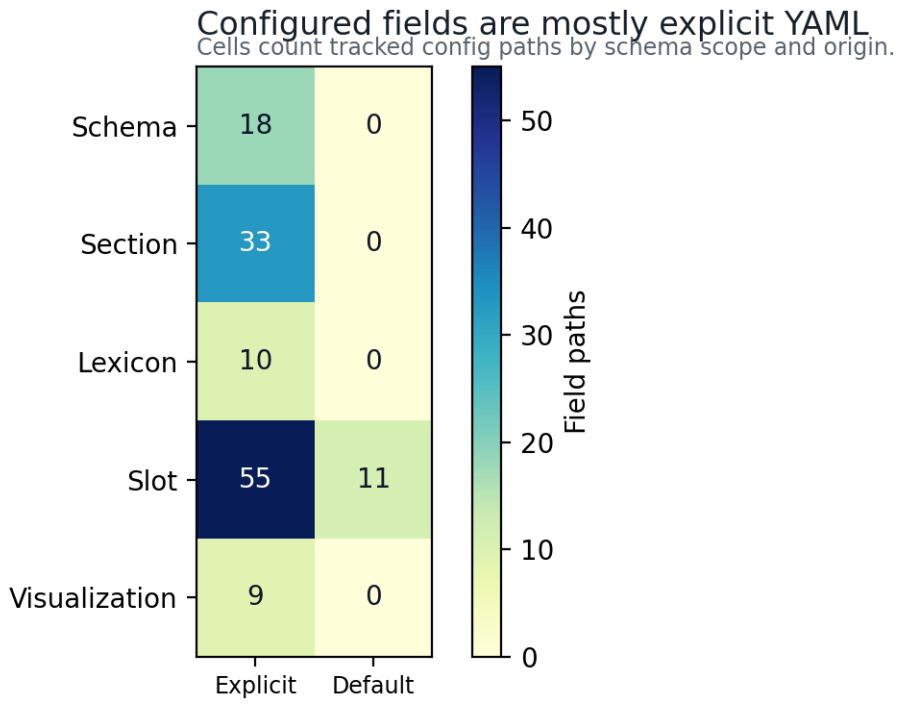


Figure 5: Configured field origin matrix

6.1 Declared Section Titles

Section key	Rendered title	Enabled
`abstract`	Abstract	True
`introduction`	Introduction: Lexicon as Data and Manuscript as Build Artifact	True
`methods`	Methods: Source-Owned Token Injection and Conditional IMRAD Assembly	True
`results`	Results: Provenance, Density, and Resolved Manuscript Surface	True
`discussion`	Discussion: Accountability Boundaries for Generated Prose	True
`configuration`	Configuration: Schema-Controlled Lexicon, Slots, and Narrative Moves	True
`evaluation`	Evaluation: Gate Criteria, QA Probes, and Failure Discovery	True
`reproducibility`	Reproducibility: Seeded Regeneration and Artifact Trace	True
`limitations`	Limitations: Non-Claims, Misuse Modes, and Human Review	True

Section key	Rendered title	Enabled
scope	Scope: Related Generators and Responsible Forking	True
authoring_contract	Authoring Contract: Human Review and Forking Obligations	True

6.2 Configuration Counts

- Seed: 431
- Composition depth: deep
- Lexicon categories: 10
- Slot rules: 22
- Token choices: 40
- Enabled sections: 11
- Method steps: 18
- Design principles: 13
- Pipeline phases: 17
- Quality probes: 14
- Authoring obligations: 8
- Explicit configured paths: 125
- Defaulted configured paths: 11
- Enabled visualization flags: 7
- Section-level configured paths: 33
- Lexicon-level configured paths: 10
- Slot-level configured paths: 66
- Narrative moves: 52
- Audit rules: 12
- Contribution claims: 9

6.3 Configured Field Summary

Measure	Count
Total tracked field paths	136
Explicit YAML paths	125
Loader-defaulted paths	11
Enabled visualization flags	7
Section-level paths	33
Lexicon-level paths	10
Slot-level paths	66
Visualization-control paths	9
Top-level schema paths	18

6.4 Configured Field Inventory

Path	Origin	Scope	Summary
madlib	explicit	schema	configured field
madlib.audit_rules	explicit	schema	12 entries
madlib.authoring_obligations	explicit	schema	8 entries
madlib.composition_depth	explicit	schema	deep
madlib.contribution_claims	explicit	schema	9 entries
madlib.design_principles	explicit	schema	13 entries
madlib.evaluation_criteria	explicit	schema	7 entries
madlib.failure_modes	explicit	schema	13 entries

Path	Origin	Scope	Summary
madlib.hypothesis	explicit	schema	Deterministic lexical injection can generate a complete conditional IMRAD manuscript while preserving token provenance, section intent, and audit-ready method evidence.
madlib.lexicon	explicit	schema	10 categories
madlib.lexicon.adjectives	explicit	lexicon	7 tokens
madlib.lexicon.artifacts	explicit	lexicon	12 tokens
madlib.lexicon.audiences	explicit	lexicon	4 tokens
madlib.lexicon.constraints	explicit	lexicon	5 tokens
madlib.lexicon.failures	explicit	lexicon	5 tokens
madlib.lexicon.measures	explicit	lexicon	8 tokens
madlib.lexicon.methods	explicit	lexicon	5 tokens
madlib.lexicon.nouns	explicit	lexicon	7 tokens
madlib.lexicon.qualities	explicit	lexicon	5 tokens
madlib.lexicon.verbs	explicit	lexicon	7 tokens
madlib.method_protocol	explicit	schema	18 entries
madlib.narrative_moves	explicit	schema	configured field
madlib.narrative_moves.abstract	explicit	section	3 moves
madlib.narrative_moves.authoring_contract	explicit	section	5 moves
madlib.narrative_moves.configuration	explicit	section	3 moves
madlib.narrative_moves.discussion	explicit	section	4 moves
madlib.narrative_moves.evaluation	explicit	section	4 moves
madlib.narrative_moves.introduction	explicit	section	4 moves
madlib.narrative_moves.limitations	explicit	section	4 moves
madlib.narrative_moves.methods	explicit	section	13 moves
madlib.narrative_moves.reproducibility	explicit	section	4 moves
madlib.narrative_moves.results	explicit	section	4 moves
madlib.narrative_moves.scope	explicit	section	4 moves
madlib.pipeline_phases	explicit	schema	17 entries
madlib.quality_probes	explicit	schema	14 entries
madlib.section_conditions	explicit	schema	configured field
madlib.section_conditions.abstract	explicit	section	enabled
madlib.section_conditions.authoring_contract	explicit	section	enabled
madlib.section_conditions.configuration	explicit	section	enabled
madlib.section_conditions.discussion	explicit	section	enabled
madlib.section_conditions.evaluation	explicit	section	enabled
madlib.section_conditions.introduction	explicit	section	enabled
madlib.section_conditions.limitations	explicit	section	enabled
madlib.section_conditions.methods	explicit	section	enabled

Path	Origin	Scope	Summary
madlib.section_conditions.reproducibility	explicit	section	enabled
madlib.section_conditions.results	explicit	section	enabled
madlib.section_conditions.scope	explicit	section	enabled
madlib.section_titles	explicit	schema	configured field
madlib.section_titles.abstract	explicit	section	Abstract
madlib.section_titles.authoring_contract	explicit	section	Authoring Contract: Human Review and Forking Obligations
madlib.section_titles.configuration	explicit	section	Configuration: Schema-Controlled Lexicon, Slots, and Narrative Moves
madlib.section_titles.discussion	explicit	section	Discussion: Accountability Boundaries for Generated Prose
madlib.section_titles.evaluation	explicit	section	Evaluation: Gate Criteria, QA Probes, and Failure Discovery
madlib.section_titles.introduction	explicit	section	Introduction: Lexicon as Data and Manuscript as Build Artifact
madlib.section_titles.limitations	explicit	section	Limitations: Non-Claims, Misuse Modes, and Human Review
madlib.section_titles.methods	explicit	section	Methods: Source-Owned Token Injection and Conditional IMRAD Assembly
madlib.section_titles.reproducibility	explicit	section	Reproducibility: Seeded Regeneration and Artifact Trace
madlib.section_titles.results	explicit	section	Results: Provenance, Density, and Resolved Manuscript Surface
madlib.section_titles.scope	explicit	section	Scope: Related Generators and Responsible Forking
madlib.seed	explicit	schema	431
madlib.slots	explicit	schema	22 slot rules, 40 token choices
madlib.slots.authoring_audience	explicit	slot	audiences -> authoring_contract (1)
madlib.slots.authoring_audience.count	defaulted	slot	1
madlib.slots.authoring_audience.section	explicit	slot	authoring_contract
madlib.slots.authoring_quality	explicit	slot	qualities -> authoring_contract (1)
madlib.slots.authoring_quality.count	defaulted	slot	1
madlib.slots.authoring_quality.section	explicit	slot	authoring_contract
madlib.slots.config_constraint	explicit	slot	constraints -> configuration (1)
madlib.slots.config_constraint.count	defaulted	slot	1
madlib.slots.config_constraint.section	explicit	slot	configuration
madlib.slots.discussion_adjective	explicit	slot	adjectives -> discussion (1)
madlib.slots.discussion_adjective.count	defaulted	slot	1

Path	Origin	Scope	Summary
madlib.slots.discussion_adjective.section	explicit	slot	discussion
madlib.slots.discussion_audience	explicit	slot	audiences -> discussion (2)
madlib.slots.discussion_audience.count	explicit	slot	2
madlib.slots.discussion_audience.section	explicit	slot	discussion
madlib.slots.evaluation_artifact	explicit	slot	artifacts -> evaluation (2)
madlib.slots.evaluation_artifact.count	explicit	slot	2
madlib.slots.evaluation_artifact.section	explicit	slot	evaluation
madlib.slots.evaluation_measure	explicit	slot	measures -> evaluation (3)
madlib.slots.evaluation_measure.count	explicit	slot	3
madlib.slots.evaluation_measure.section	explicit	slot	evaluation
madlib.slots.intro_nouns	explicit	slot	nouns -> introduction (4)
madlib.slots.intro_nouns.count	explicit	slot	4
madlib.slots.intro_nouns.section	explicit	slot	introduction
madlib.slots.intro_verbs	explicit	slot	verbs -> introduction (4)
madlib.slots.intro_verbs.count	explicit	slot	4
madlib.slots.intro_verbs.section	explicit	slot	introduction
madlib.slots.limitation_failure	explicit	slot	failures -> limitations (3)
madlib.slots.limitation_failure.count	explicit	slot	3
madlib.slots.limitation_failure.section	explicit	slot	limitations
madlib.slots.method_artifact	explicit	slot	artifacts -> methods (2)
madlib.slots.method_artifact.count	explicit	slot	2
madlib.slots.method_artifact.section	explicit	slot	methods
madlib.slots.method_constraint	explicit	slot	constraints -> methods (1)
madlib.slots.method_constraint.count	defaulted	slot	1
madlib.slots.method_constraint.section	explicit	slot	methods
madlib.slots.method_name	explicit	slot	methods -> methods (1)
madlib.slots.method_name.count	defaulted	slot	1
madlib.slots.method_name.section	explicit	slot	methods
madlib.slots.method_quality	explicit	slot	qualities -> methods (2)
madlib.slots.method_quality.count	explicit	slot	2
madlib.slots.method_quality.section	explicit	slot	methods
madlib.slots.reproducibility_artifact	explicit	slot	artifacts -> reproducibility (2)
madlib.slots.reproducibility_artifact.count	explicit	slot	2

Path	Origin	Scope	Summary
madlib.slots.reproducibility_artifact.section	explicit	slot	reproducibility
madlib.slots.result_artifact	explicit	slot	artifacts -> results (2)
madlib.slots.result_artifact.count	explicit	slot	2
madlib.slots.result_artifact.section	explicit	slot	results
madlib.slots.result_measure	explicit	slot	measures -> results (3)
madlib.slots.result_measure.count	explicit	slot	3
madlib.slots.result_measure.section	explicit	slot	results
madlib.slots.scope_audience	explicit	slot	audiences -> scope (1)
madlib.slots.scope_audience.count	defaulted	slot	1
madlib.slots.scope_audience.section	explicit	slot	scope
madlib.slots.scope_constraint	explicit	slot	constraints -> scope (1)
madlib.slots.scope_constraint.count	defaulted	slot	1
madlib.slots.scope_constraint.section	explicit	slot	scope
madlib.slots.study_adjective	explicit	slot	adjectives -> abstract (1)
madlib.slots.study_adjective.count	defaulted	slot	1
madlib.slots.study_adjective.section	explicit	slot	abstract
madlib.slots.study_noun	explicit	slot	nouns -> abstract (1)
madlib.slots.study_noun.count	defaulted	slot	1
madlib.slots.study_noun.section	explicit	slot	abstract
madlib.slots.study_verb	explicit	slot	verbs -> abstract (1)
madlib.slots.study_verb.count	defaulted	slot	1
madlib.slots.study_verb.section	explicit	slot	abstract
madlib.visualizations	explicit	visualization	enabled
madlib.visualizations.configured_field_matrix	explicit	visualization	true
madlib.visualizations.enabled	explicit	visualization	true
madlib.visualizations.field_origin_summary	explicit	visualization	true
madlib.visualizations.provenance_trace_map	explicit	visualization	true
madlib.visualizations.quality_gate_matrix	explicit	visualization	true
madlib.visualizations.section_configuration_heatmap	explicit	visualization	true
madlib.visualizations.section_token_allocation	explicit	visualization	true
madlib.visualizations.token_injection_flow	explicit	visualization	true

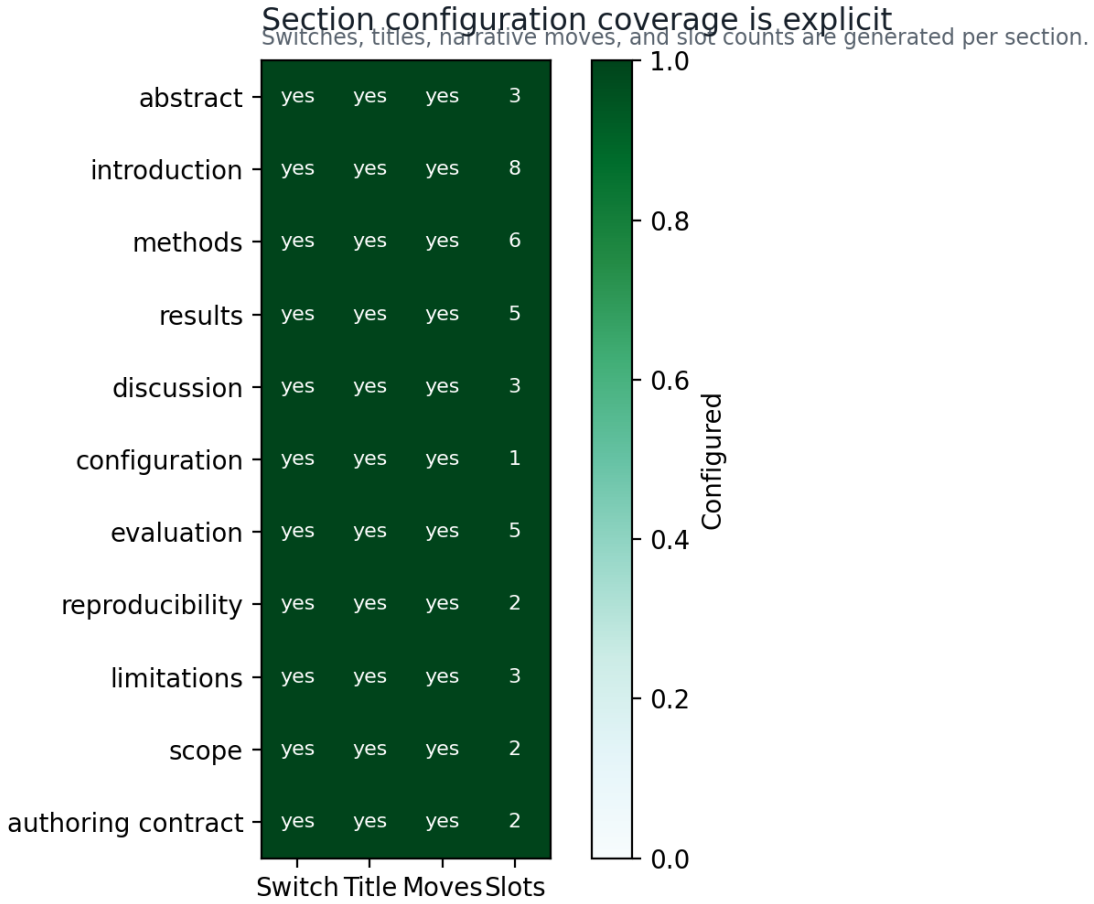


Figure 6: Section configuration heatmap

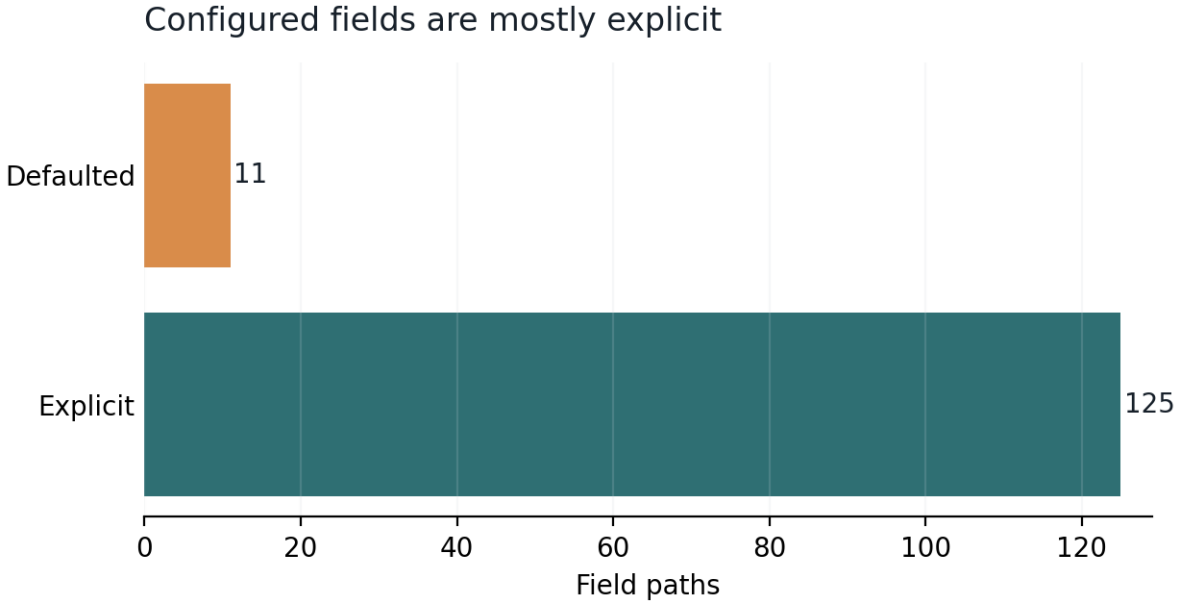


Figure 7: Field origin summary

7 Evaluation: Gate Criteria, QA Probes, and Failure Discovery

The evaluation section is configured to name readiness criteria, connect criteria to artifacts, separate local checks from publication readiness, and make failure probes visible. The local readiness surface is not a human preference score; it is a set of deterministic checks that connect generated manuscript claims to source files and pipeline gates.

The active criteria use analysis, copy, pytest, validation as gate labels and inspect artifacts such as token-injection flow, quality-gate matrix, configured-field figures. A passing run means the exemplar is locally render-ready: placeholders resolve, token provenance is present, figure references are registered, evidence scanning has not found unsupported numbers, and project design overlays remain internally consistent.

That readiness is deliberately narrower than publication readiness. A local pass does not imply a standalone DOI, external release, reader preference result, or empirical validation. It means the tracked project tree can regenerate the committed artifact surface through its declared pipeline.

The QA probes are Method row completeness, Field-origin visibility, Placeholder survival, Provenance completeness, Section-switch observability, Figure registry coverage, Method-figure alignment, Evidence cleanliness, Fork readiness, Copied-output parity, Digest invariant review, Claim-ledger alignment, Review packet completeness, Fork migration sufficiency. They are phrased as questions so they can be reused by reviewers and by forks of the exemplar: did the placeholder disappear, did the provenance survive, did the figure registry cover every reference, and did copied outputs preserve the same evidence surface that validation inspected?

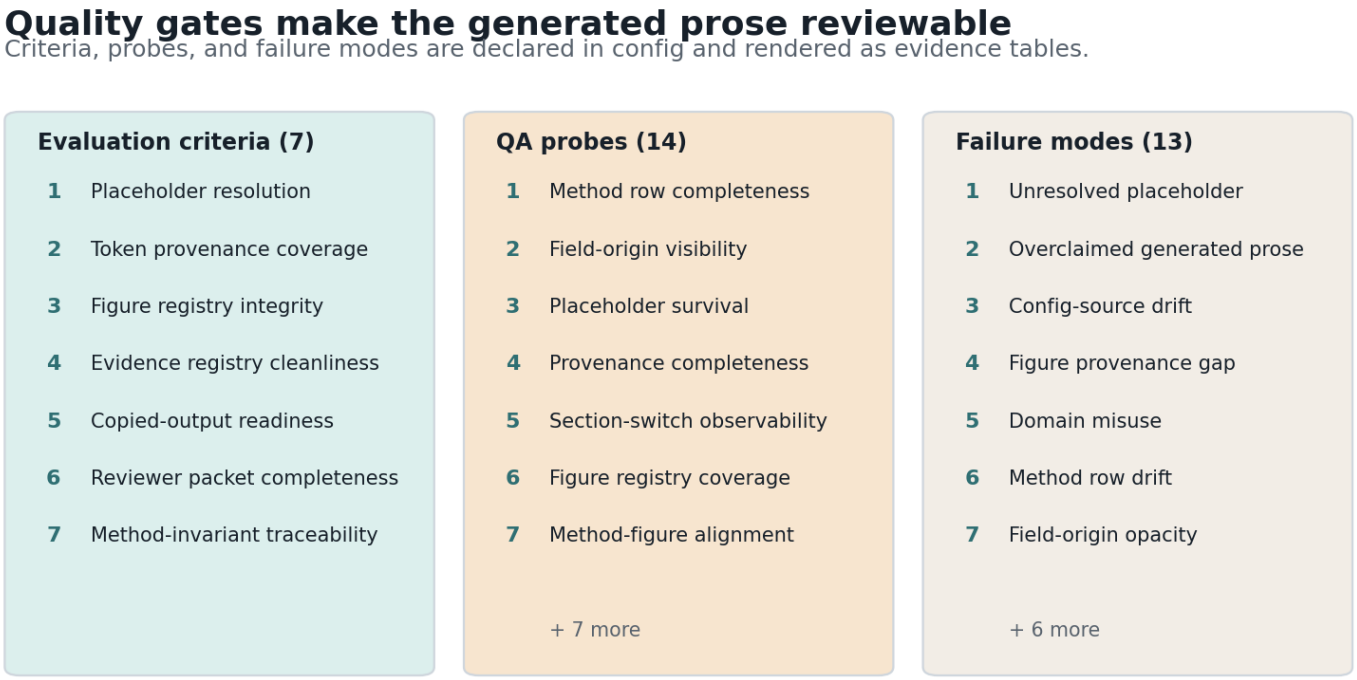


Figure 8: Quality gate matrix

7.1 Evaluation Criteria

Criterion	Target	Evidence	Gate
Placeholder resolution	No unresolved uppercase manuscript placeholders remain in output/manuscript or rendered web output.	tests/test_manuscript_variables.py and rg unresolved-token scan.	`pytest`
Token provenance coverage	Every selected token maps to category, selected value, section, and config pointer.	output/reports/injection_trace.json and output/data/token_inventory.json.	`analysis`

Criterion	Target	Evidence	Gate
Figure registry integrity	Every referenced figure label is present in ../figures/figure_registry.json.	scripts/04_validate_output.py figure registry check.	validation
Evidence registry cleanliness	Generated manuscript numbers and claims pass the project evidence registry.	output/reports/evidence_registry_audit	validation
Copied-output readiness	PDF, HTML, slides, figures, data, and reports copy into output/templates/template_madlib.	scripts/05_copy_outputs.py output statistics.	copy
Reviewer packet completeness	Hydrated Markdown, rendered PDF, web output, slides, figures, data, reports, validation results, and copy statistics are all present for review.	Stage 04 validation report and Stage 05 output_statistics.json.	copy
Method-invariant traceability	Token choices can be explained only by seed, slot, category, ordinal, and ordered category inventory.	tests/test_tokens.py and generated Methods digest prose.	pytest

7.2 Quality Probes

Probe	Question	Passing signal	Artifact
Method row completeness	Does the protocol table cover schema intake, token planning, composition, figures, validation, copy, and review handoff?	method_protocol includes rows for every major pipeline responsibility.	manuscript/config.yaml and output/data/section_plan.json
Field-origin visibility	Can a reviewer tell which visible fields were authored and which were defaulted?	Configured-field inventory and summary tables report explicit and defaulted paths.	output/data/configured_field_inventory.json
Placeholder survival	Did any source token survive hydration?	No uppercase placeholders are found in generated manuscript or web files.	output/manuscript and output/web
Provenance completeness	Can every selected token be traced to a category, section, value, and config key?	The injection trace and token inventory contain one row for each generated token.	output/reports/injection_trace.json
Section-switch observability	Does a disabled section resolve to a visible explanation?	Disabled section bodies cite their controlling section condition.	output/data/section_plan.json
Figure registry coverage	Does every manuscript figure reference have a generated registry entry?	Figure registry validation passes.	../figures/figure_registry.json
Method-figure alignment	Do method figures describe generated data rather than decorative or unsupported claims?	Figure registry captions, nonblank PNG tests, and manual visual QA align with artifact data.	output/figures
Evidence cleanliness	Do generated claims stay within local evidence boundaries?	Evidence registry validation passes without unsupported claims.	output/reports/evidence_registry.json
Fork readiness	Does the authoring contract tell downstream forks what to extend before adding domain claims?	Authoring obligations cite config diffs, claim ledger updates, validators, and full reruns.	output/manuscript/10_authoring_contract.md
Copied-output parity	Did copied deliverables preserve the validated project output surface?	Copy-stage statistics include PDF, HTML, slides, figures, data, and reports.	output/templates/template_madlib

Probe	Question	Passing signal	Artifact
Digest invariant review	Are the allowed token-selection inputs documented and protected by tests?	Methods prose names the digest inputs and token tests prove seed/category sensitivity.	`src/tokens.py and output/manuscript/02_methodology.md`
Claim-ledger alignment	Do method and documentation claims point to config, source, generated artifacts, or explicit non-claim boundaries?	Claim-ledger rows cover expanded method protocol and fork-validator boundaries.	`data/claim_ledger.yaml`
Review packet completeness	Can a reviewer inspect every output surface needed to audit the method?	Copied outputs include manuscript, web, slides, figures, data, reports, validation, and copy statistics.	`output/templates/template_madlib and output/reports/output_statistics.json`
Fork migration sufficiency	Does the documentation tell forks which surfaces to change before adding domain claims?	README, STANDALONE, manuscript README, and Authoring Contract list config, source, test, validator, and claim-ledger obligations.	`README.md, STANDALONE.md, manuscript/README.md, and output/manuscript/10_authoring_contract.md`

8 Reproducibility: Seeded Regeneration and Artifact Trace

Re-running generation with seed `431` and the same lexicon produces the same token plan. The artifact set records `token_inventory.json`, `section_plan.json`, `injection_trace.json`, `manuscript_variables.json`, `figure_registry.json`, and the cover/results/configuration/evaluation figure set so the manuscript can be audited without reading the PDF.

The protocol emits MadlibConfig, review scenario, explicit/default path inventory, validated lexicon, digest input records, selection invariant set, TokenPlan, enabled section set, section variables, Markdown evidence tables, claim-aligned evidence surface, registered figure set, output/data, output/reports, and output/figures, output/manuscript, validated project output, review packet, copied publication-review bundle, fork migration notes. Project tests cover deterministic token choice, seed sensitivity, category-input sensitivity, malformed config rejection, section disablement, artifact writing, and unresolved manuscript-token detection. The shared output validator then checks rendered PDFs, Markdown, figure registry, evidence registry, and design overlays.

The copied root output is therefore a consequence of local source and config. Generated files remain disposable; the durable contract is the ability to regenerate them from the tracked project tree and to observe the same validation gates passing.

- Config hash: `a37b58a23935b5ed`
- Generated: `2026-06-21T16:44:33Z`
- Python: `3.12.13`
- Platform: `Darwin arm64`

9 Limitations: Non-Claims, Misuse Modes, and Human Review

The limitations section is configured to state non-claims, identify misuse modes, preserve human review, and require domain validators for domain claims. The central limitation is that deterministic token injection can make manuscript assembly auditable, but it cannot make a weak claim true or a missing source appear.

The declared failure modes are Unresolved placeholder, Overclaimed generated prose, Config-source drift, Figure provenance gap, Domain misuse, Method row drift, Field-origin opacity, Visual-method mismatch, Fork without validators, Digest invariant drift, Claim ledger omission, Review packet incompleteness, Fork migration ambiguity. They are included in the manuscript because this template is meant to teach the boundary, not hide it. A useful fork should extend this table when it adds domain-specific claims, validators, or publication targets.

The author remains responsible for theory, citations, reader expectations, and domain evidence. The generator can enforce structure and provenance; it cannot supply judgment. That division is the main safety property of the exemplar.

9.1 Failure Modes

Failure mode	Risk	Detection	Mitigation
Unresolved placeholder	A source Markdown token is added without a generated variable.	Project test scans output/manuscript and rendered web output for uppercase placeholders.	Add variable generation, tests, and rerun z_generate_manuscript_variables. before render.
Overclaimed generated prose	Generated text implies a standalone DOI, empirical validation, or external release that does not exist.	Claim ledger review, publication metadata review, and evidence registry validation.	Keep publication fields blank and use local-only claim boundaries until evidence exists.
Config-source drift	Documentation names a schema feature that source code no longer parses.	Config tests instantiate the schema and docs point to generated tables.	Update config loader, docs, and tests together.
Figure provenance gap	A figure is referenced in manuscript output without a registry entry.	Stage 04 figure registry validation.	Emit figure_registry.json from src.analysis before rendering.
Domain misuse	A fork treats lexical substitution as evidence for a domain-specific research claim.	Failure-mode table, scope text, and downstream domain validators.	Add domain-specific data, validators, and claim ledgers before making domain claims.
Method row drift	The Methods prose describes a protocol row or phase that no longer exists in config.	Composition tests, summary report review, and generated method tables.	Update method_protocol, pipeline_phases, composition prose, and tests together.
Field-origin opacity	A rendered field appears configured even though it came from a loader default.	configured_field_inventory.json and configured-field summary figures.	Expose explicit/default counts in the manuscript and review optional defaults before release.
Visual-method mismatch	A figure implies a method claim that is not backed by generated data or registry metadata.	Figure registry validation, nonblank figure tests, and manual visual QA.	Generate figures only from config, TokenPlan, and inventory data, then rerun validation.
Fork without validators	A downstream project changes vocabulary or claims but keeps only the exemplar's generic gates.	Authoring contract review and claim ledger review.	Add domain validators, domain evidence artifacts, and claim-ledger entries before asserting domain findings.
Digest invariant drift	A future edit lets renderer state, file order, or ambient prose influence token selection.	Seed-stability, category-sensitivity, and method-invariant tests.	Keep token selection isolated in src/tokens.py and document allowed digest inputs in Methods.
Claim ledger omission	A generated claim appears in prose or documentation without a matching local evidence row or non-claim boundary.	Claim ledger review, evidence registry validation, and documentation review.	Add claim-ledger rows or remove the unsupported claim before rendering copied outputs.

Failure mode	Risk	Detection	Mitigation
Review packet incompleteness	A reviewer receives PDF or HTML without the data, reports, figures, validation output, or copy statistics needed to inspect the method.	Stage 05 output statistics and copied-output validation.	Regenerate Stages 02-05 and include the full copied output surface in review.
Fork migration ambiguity	A fork leaves authors unsure which exemplar surfaces must change for a domain-specific manuscript.	README, STANDALONE notes, manuscript README, and Authoring Contract review.	Document required config, source, test, claim-ledger, validator, and pipeline changes before making domain claims.

10 Scope: Related Generators and Responsible Forking

The exemplar is a pipeline testbed, not a natural-language quality benchmark. It covers deterministic token selection, conditional section bodies, section-title injection, provenance tables, and generated evidence artifacts. It does not evaluate semantic originality, factual truth beyond the local configuration, or reader preference.

The configured scope moves are distinguish generation from truth, limit publication claims, point to local evidence, and explain responsible forking. The closest related systems are not general-purpose chatbots but source-owned report generators, literate-programming documents, static-site data templates, and reproducible manuscript pipelines. `template_madlib` contributes a deliberately small version of that idea for research manuscripts.

Publication metadata remains conservative. The local `CITATION.cff`, `.zenodo.json`, and `codemeta.json` describe this exemplar inside the shared template repository; they do not claim a live standalone DOI, external release, or empirical validation outside the generated local artifacts.

11 Authoring Contract: Human Review and Forking Obligations

The authoring contract is configured to state human responsibilities, name fork obligations, connect review to generated evidence, require domain validators before domain claims, and document fork migration notes. It addresses pipeline maintainers directly because the generator can preserve structure, provenance, and reviewability, but it cannot decide what a field should claim. Human authors remain responsible for theory, interpretation, citations, and the choice to publish.

The declared obligations are Review generated claims, Review config diffs, Extend claim evidence, Add domain validators, Rerun the full project path, Review method invariants, Assemble reviewer packet, Write fork migration notes. They convert responsible use into a checklist: review generated claims in the hydrated manuscript, inspect the generated evidence tables, extend the claim ledger when new claims appear, and add domain-specific validators before the template is used for a real empirical or theoretical manuscript.

The quality standard is claim humility. A fork that only changes words has not preserved the exemplar. A responsible fork changes the config, adjusts source-owned composition where necessary, regenerates artifacts, and reruns the same tests and validation gates before treating the output as reader-ready.

11.1 Authoring Obligations

Obligation	Required action	Review surface
Review generated claims	Inspect hydrated manuscript bodies before copied outputs are treated as reader-ready.	`output/manuscript and output/web`
Review config diffs	Treat lexicon, slot, title, move, and section-switch edits as source-data changes.	`manuscript/config.yaml`
Extend claim evidence	Update the claim ledger when generated prose adds a new claim boundary.	`data/claim_ledger.yaml`
Add domain validators	Add tests and validation artifacts before using the template for domain-specific claims.	`tests and output/reports`
Rerun the full project path	Regenerate analysis artifacts, render outputs, validate outputs, and copy deliverables.	`pipeline command logs`
Review method invariants	Confirm changed tokens are explained only by seed, slot, category, ordinal, or category inventory changes.	`src/tokens.py, token_inventory.json, and output/manuscript/02_methodology.md`
Assemble reviewer packet	Provide hydrated manuscript, rendered outputs, figures, data, reports, validation report, and output statistics together.	`output/templates/template_madlib`
Write fork migration notes	Document config, source, test, validator, and claim-ledger changes required by a domain fork.	`README.md, STANDALONE.md, and data/claim_ledger.yaml`

References