Symbolic Correspondence Generation: A Formal Specification with Scoped Certificates and Reproducible Batch Semantics

Flyxion
Independent Researcher

November 2026

Abstract

This paper gives a formal specification of a symbolic system that produces individualized email drafts from a versioned memory of projects, author preferences and recipient histories. The system separates a message plan, which records why the author is writing, which facts may be stated and what response is requested, from a rendered draft, which expresses that plan in prose. Planning is governed by a small declarative evaluator with three-valued logic in which missing evidence is unknown rather than false, so that a positive requirement can never be met by the absence of a contradiction. Every assertion a draft may make, including pragmatic implications such as a prior exchange, must be backed by a scope-carrying certificate that records its sources, the inference rule used, the recipient and occasion scope, and whether the underlying evidence may be mentioned or only used for planning. Realization operates on typed clause plans under a validity predicate that rejects unsupported implications, unrecoverable referents, repeated discourse moves, a second request, a changed action, and violations of layered author preferences. We state and prove the properties that make the design checkable: monotonicity of the evaluator on a defined fragment, well-definedness of preference resolution, soundness of certificate coverage relative to declared and lexically detected implications, stability of candidate selection under tightened constraints, and independence of batch output from worker count, chunking and interruption under a recorded engine identity. We define three separate diversity measures, a capacity model for distinctness-constrained generation, and an append-only hash-chained ledger with an accounting identity. We then report measurements from a reference implementation: a 10,008 slot experiment, a 100,008 slot run, and a capacity estimate obtained by sampling the grammar's path space. We state what remains unverified, including the million slot benchmark, and identify the limits of the lexical implication check, the recency sensitivity of bound evidence, and the bounded-sample source of the exhaustion events observed at scale.

Contents

1 Introduction and Scope **4**

2 Data Model and Snapshots **5**
2.1 Notation 5
2.2 Records 5
2.3 Snapshots 6
2.4 Well-Formedness 6

3 Evidence Logic **7**
3.1 Syntax 7
3.2 Semantics 7
3.3 Monotonicity and Its Limits 8

4 Preference Resolution **9**

5 Planning **10**
5.1 Candidate Pairs 10
5.2 Scoring 11
5.3 Selection 11
5.4 Abstention 12
5.5 The Plan Object 12

6 Certificates and the Assertion Calculus **12**
6.1 Certificates 12
6.2 Construction 13
6.3 Aspect Relegation 14

7 Surface Realization **14**
7.1 Rules 14
7.2 Eligibility 15
7.3 Discourse Orders and Candidate Identifiers 15
7.4 Validity 15
7.5 Properties 16
7.6 Candidate Generation and Fallback 17

8 Determinism and Reproducibility **17**
8.1 Seeds and Selection 17
8.2 Execution Independence 18
8.3 Engine Identity on Resume 19

9 Diversity, Capacity and Distinctness **19**
9.1 Three Separate Measures 19
9.2 Distinctness Constraints and Capacity 20

10 Ledger, Storage and Audit **21**
10.1 Slot States and Events 21
10.2 Hash Chain 21
10.3 Accounting 22
10.4 Storage Layout 22
10.5 Export 23

11 Offline Maintenance **23**

12 Acceptance Criteria **24**

13 Reference Implementation and Measurements **26**
13.1 Structure 26
13.2 Corpus 26
13.3 Measurements 26
13.4 Million-Slot Benchmark 27

14 Limitations and Open Problems **27**

A Grammar Rule Schema and Inventory **28**
A.1 Rule Schema 28
A.2 Inventory 28

B Algorithms **29**

C Command-Line Interface **30**

1 Introduction and Scope

A system that drafts individual correspondence faces a tension that is easy to state and hard to enforce. The drafts should differ because the recipients differ, which means they should draw on evidence about those recipients and make requests that fit what the author actually wants. Yet every sentence that refers to a recipient's history is a claim, and a fluent sentence that claims more than the record supports is worse than a generic one. A message that opens by thanking someone again presupposes an earlier thanks. A message that follows up on an exchange presupposes the exchange. A message that mentions a comment on page 14 presupposes that the comment exists and that the author may say so. The specification studied here treats these presuppositions as first-class objects and builds the generator around the requirement that none of them may be introduced by wording alone.

The design rests on a separation of two representations. The *message plan* is a structured object that records the reason for writing, the project concerned, the set of propositions that may be asserted, the single response requested, and the aspects of the project that are foregrounded, optional or excluded. The *rendered draft* is prose produced from the plan by a grammar. Variation in expression is permitted only within the bounds that the plan fixes. Variation in the plan itself, meaning a different purpose, different evidence or a different request, is a different kind of variation and is measured separately from variation in wording. A recipient record that contains only a name can support a salutation, and the system records that as name substitution, not as individualization.

Contributions. This paper formalizes the specification and states the properties it is meant to guarantee. Section 2 fixes the data model and the notion of a frozen snapshot. Section 3 defines the evidence logic and proves a monotonicity result on a bounded fragment, together with an explicit account of where monotonicity fails. Section 4 defines layered preference resolution and shows that it is a well-defined function. Section 5 defines planning, scoring and abstention. Section 6 defines scope-carrying certificates and the assertion calculus, and Section 7 defines realization and proves that validity entails certificate coverage. Section 8 treats determinism, proving stability of selection and independence of batch output from execution schedule. Section 9 defines the diversity measures and a capacity model, Section 10 the ledger and accounting, Section 11 the offline maintenance cycle, and Section 12 the acceptance criteria. Section 13 reports measurements from a reference implementation, and Section 14 states its limits.

Status of the claims. Throughout, a statement is marked as a definition when it fixes the specification, as a theorem or proposition when it is proved from the definitions, and as a measurement when it comes from a run of the reference implementation (version 0.1.0, schema version 1). The reference implementation is deterministic Python with SQLite storage and a Bash command interface. Its test suite contains 33 tests, all of which pass. A million slot benchmark is defined but has not been executed, and no claim in this paper depends on its outcome.

2 Data Model and Snapshots

2.1 Notation

Let $\mathbb{D}$ denote the set of calendar dates with its usual total order. A reference date $d^{*} \in \mathbb{D}$ is supplied explicitly with every run, and no component of the system reads a wall clock when making a decision. Let H denote SHA-256 [4] and $\mathrm{canon}(x)$ denote the canonical serialization of a JSON-like value x, with object keys sorted, no insignificant whitespace, and characters left unescaped. For a finite sequence of strings $a_1, \dots, a_n$ let

$$h(a_1, \dots, a_n) \;=\; \text{the first 64 bits of } \mathrm{H}(a_1 \,\|\, \text{0x1F} \,\|\, a_2 \,\|\, \cdots \,\|\, a_n),$$

read as an unsigned integer. All pseudo-random choices in the system are values of h applied to a master seed, a slot identifier, a stage name and a choice identifier. For a finite non-empty sequence L we write $\mathrm{pick}(L;\, a_1, \dots, a_n) = L\big[h(a_1, \dots, a_n) \bmod |L|\big]$.

2.2 Records

The editable memory consists of the following record types. Each has a stable identifier that is unique within its type.

Definition 2.1 (Project)**.** A project is a tuple $\pi = (\mathrm{id}, \mathrm{title}, \mathrm{short}, \mathrm{url}, \mathrm{status}, \mathrm{kind}, \mathrm{age}, \mathrm{rev}, \mathrm{revdate}, \mathrm{asp})$ where $\mathrm{status} \in \{\mathrm{revising}, \mathrm{published}, \mathrm{planned}\}$, $\mathrm{kind} \in \{\mathrm{essay}, \mathrm{monograph}\}$, and asp is a partial map from aspect names to text. The aspect names in use are title, url, argument, revision_note and defect_kinds, the last being a non-empty list of strings.

Definition 2.2 (Recipient)**.** A recipient is a tuple $r = (\mathrm{id}, \mathrm{name}, \mathrm{given}, \mathrm{lang}, \mathrm{ref})$ with a language tag and an optional address reference. The address itself is never part of memory. It is resolved only at export, from a separate file keyed by the reference.

Definition 2.3 (Evidence)**.** An evidence record is $e = (\mathrm{id}, s, p, j, v, \ell, \mathrm{src}, t, U, \nu)$ with subject s a recipient identifier, predicate $p \in \mathcal{P}$, optional project j, optional value v and locator ℓ, a source description, an observation date $t \in \mathbb{D}$, an allowed-use set $U \subseteq \{\mathsf{plan}, \mathsf{mention}\}$, and a negation flag $\nu \in \{0, 1\}$. The predicate set is

$$\begin{aligned}\mathcal{P} = \{&\mathrm{downloaded}, \mathrm{read}, \mathrm{commented_on}, \mathrm{offered_to_help_with},\\ &\mathrm{requested_revision}, \mathrm{received_revision}, \mathrm{corresponded_with}, \mathrm{asked_not_to_contact}\}.\end{aligned}$$

The first four predicates are those distinguished by the specification. The remaining four are required by the three declared purposes and by the abstention cases. An implication between predicates exists only if a rule says so (Definition 2.5). In particular, a record of the predicate commented_on with a locator supports a claim about that comment and, through a stated inference rule, a claim that the passage was read. It does not establish that a whole work was read.

Definition 2.4 (Goal)**.** A goal is $g = (\mathrm{id}, \mathrm{purpose}, \Pi_g, \mathrm{act}, \mathrm{prio}, [a, b], F, O, X, \mathrm{req}, \mathrm{dis})$ where Π_g is a list of project identifiers, act is the action requested of the recipient, $\mathrm{prio} \in \mathbb{N}$ is the author's priority for the goal, $[a, b]$ is an activity window, F, O, X are disjoint sets of aspect

names (foregrounded, optional, excluded), and req and dis are expressions of the evidence logic of Section 3. The disqualifier dis may be absent, in which case it is treated as the constant false.

Definition 2.5 (Inference rule)**.** An inference rule is a triple (id, k, k') of certificate kinds with an optional flag requiring a locator. It licenses a certificate of kind k' from a certificate of kind k with the same sources and the same use.

Definition 2.6 (Preference)**.** A preference is $q = (\mathrm{id}, \sigma, \kappa, \mathrm{val}, \eta, \mathrm{prio}, W)$ with scope $\sigma \in \{\mathrm{global}\} \cup \{\mathrm{project}{:}j\} \cup \{\mathrm{occasion}{:}o\}$, key κ, value, hardness flag $\eta \in \{0, 1\}$, integer priority, and an optional effective window $W \subseteq \mathbb{D}$.

Definition 2.7 (Occasion and slot)**.** An occasion is a pair $(o, \mathrm{allowed}_o)$ with a set of permitted purposes and a flag needr_o stating whether a recipient-specific reason is required. A draft slot is $s = (\mathrm{id}, \mathrm{mode}, r, o, j?, p?)$ with mode in $\{\mathrm{correspondence}, \mathrm{experiment}\}$, a recipient, an occasion, and optional constraints on project and purpose. At most one draft is emitted per slot.

Grammar rules and the trigger lexicon are records of the memory as well, and are defined in Section 7. A batch manifest is a record of the form $\mathcal{B} = (\mathrm{id}, \mathrm{mode}, \mathrm{snapshot}, \mathrm{seed}, d^*, K, \beta, \mathrm{distinct}, \mathrm{slots})$ where $K \in \mathbb{N}$ is the number of candidates sampled per slot, $\beta \in \mathbb{N}$ is the score band, and distinct states whether distinct normalized text is required.

2.3 Snapshots

Definition 2.8 (Snapshot)**.** Let $\mathcal{F}$ be the set of files of the memory directory, excluding derived files, and let $f \mapsto \mathrm{H}(f)$ map each file to its digest. A snapshot is the pair $(\mathcal{M}, \mathrm{hash})$ where $\mathcal{M}$ is the parsed memory and $\mathrm{hash} = \mathrm{H}\big(\mathrm{canon}(\{(\mathrm{path}(f), \mathrm{H}(f)) : f \in \mathcal{F}\})\big)$.

A batch reads only a snapshot, never the live memory. Loading a snapshot recomputes the digest and fails if it differs from the recorded one, so an edit to a project title made midway through generation has no effect on the batch and a tampered snapshot is detected. The SQLite database built at freeze time is a projection of the snapshot and is rebuilt from it, so it is not a source of truth.

2.4 Well-Formedness

Definition 2.9 (Valid memory)**.** A memory $\mathcal{M}$ is valid when the following hold. Identifiers are unique within each record type. Every reference resolves: evidence subjects name recipients, evidence and goal projects name projects, slot recipients, occasions, projects and purposes exist, and inference and trigger kinds belong to the kind set of Section 6. Every predicate that occurs in a goal condition belongs to $\mathcal{P}$, and every operation belongs to the operation set of Section 3. Preference resolution is conflict-free at every checked point (Proposition 4.4). Every grammar rule uses known slots, placeholders, assertion kinds and referents, request rules declare an action and obligation and count exactly one request, non-request rules count none, and every rule that contains a pronoun or definite reference either declares the referent it needs or contains a title placeholder. Finally, for every goal and every required slot there is at least one usable rule and at least one usable canonical rule.

The last condition guarantees that the canonical fallback of Section 7 exists for each purpose. Validation reports every violated condition and does not stop at the first.

3 Evidence Logic

3.1 Syntax

Goal conditions are written in a deliberately small language so that every planning decision can be inspected. The language has a leaf form that queries evidence, comparison forms over resolved operands, and the Boolean connectives.

Definition 3.1 (Expressions)**.** Let Σ be the set of evidence specifications $\varsigma = (p, \mathrm{pf}, \mathrm{vin}, \mathrm{veq}, \mathrm{use}, \mathrm{age}, \mathrm{bind})$ with predicate $p \in \mathcal{P}$, optional project filter, optional value-membership list, optional value-equality operand, a use requirement $\mathrm{use} \in \{\mathsf{plan}, \mathsf{mention}\}$ defaulting to plan, an optional maximum age in days, and an optional bind name. Expressions are generated by

$$\begin{aligned}\varphi \;::=\; & \mathsf{has}(\varsigma) \mid \mathsf{missing}(\varsigma) \mid \mathsf{eq}(a,b) \mid \mathsf{in}(a,L) \mid \mathsf{cmp}_{\bowtie}(a,b) \mid \\ & \mathsf{all}(\varphi_1,\ldots,\varphi_n) \mid \mathsf{any}(\varphi_1,\ldots,\varphi_n) \mid \mathsf{not}(\varphi)\end{aligned}$$

with $\bowtie \in \{<, \leq, >, \geq\}$ comparing dates. Operands a, b are literals or references. A reference is \$project, \$project.f, \$goal.f, \$ref_date, or \$$x.f$ for a bind name x.

3.2 Semantics

The value set is $\mathbf{K}_3 = \{\mathsf{F}, \mathsf{U}, \mathsf{T}\}$, with the truth order $\mathsf{F} < \mathsf{U} < \mathsf{T}$ and the information order $\sqsubseteq$ in which $\mathsf{U} \sqsubseteq \mathsf{T}$ and $\mathsf{U} \sqsubseteq \mathsf{F}$ while T and F are incomparable. The connectives are the strong Kleene connectives [1]: conjunction is minimum and disjunction is maximum in the truth order, and negation exchanges T and F and fixes U.

An evaluation context is $c = (E, \pi, g, d^*, \beta)$ where E is the list of evidence records of one recipient, π is a project, g is a goal, and β is a partial map from bind names to evidence records. Context evaluation is stateful in β only, and the state flows from left to right.

Definition 3.2 (Matching)**.** A record e matches ς in c when $e.p = \varsigma.p$; the project filter, if not the wildcard, equals $e.j$ after resolving references; $e.v$ is a case-insensitive member of ς.vin when that list is given; $e.v$ equals the resolved ς.veq when that operand is given; $\varsigma.\mathrm{use} \in e.U$; and, when an age bound is given, $0 \leq d^* - e.t \leq \varsigma.\mathrm{age}$. Let $M^+(\varsigma, c)$ be the matching records with $\nu = 0$ and $M^-(\varsigma, c)$ those with $\nu = 1$.

Definition 3.3 (Evaluation)**.** The value $[\![\varphi]\!](c) \in \mathbf{K}_3$ is defined as follows.

$$[\![\mathsf{has}(\varsigma)]\!] = \begin{cases} \mathsf{T} & \text{if } M^+ \neq \emptyset, \text{ and the record maximal in } (t, \mathrm{id}) \text{ is bound to } \varsigma.\mathrm{bind}, \\ \mathsf{F} & \text{if } M^+ = \emptyset \text{ and } M^- \neq \emptyset, \\ \mathsf{U} & \text{otherwise,} \end{cases}$$

$$[\![\mathsf{missing}(\varsigma)]\!] = \begin{cases} \mathsf{T} & \text{if } M^+ = \emptyset, \\ \mathsf{F} & \text{otherwise,} \end{cases}$$

$$[\![\mathsf{eq}(a,b)]\!] = \mathsf{U} \text{ if either operand is unresolved, else } \mathsf{T} \text{ iff the operands are equal,}$$

$$[\![\mathsf{in}(a,L)]\!] = \mathsf{U} \text{ if either operand is unresolved, else } \mathsf{T} \text{ iff } a \in L,$$

$$[\![\mathsf{cmp}_{\bowtie}(a,b)]\!] = \mathsf{U} \text{ if either operand is unresolved, else } \mathsf{T} \text{ iff } a \bowtie b \text{ as dates,}$$

$$[\![\mathsf{all}(\varphi_1,\ldots,\varphi_n)]\!] = \min_i [\![\varphi_i]\!], \qquad [\![\mathsf{any}(\varphi_1,\ldots,\varphi_n)]\!] = \max_i [\![\varphi_i]\!],$$

$$[\![\mathsf{not}(\varphi)]\!] = \neg [\![\varphi]\!].$$

The arguments of all and any are evaluated in order and without short-circuiting, so every bind established by an earlier argument is visible to later arguments.

The central design choice is visible in the first clause. When no record matches a has query the value is U, never F. A value of F requires an explicit negated record. The operation missing is the only two-valued query, and it is the only way for a rule to state that the absence of a record is itself a condition.

Definition 3.4 (Eligibility of a goal-project pair)**.** For a context c whose goal has requirement req and disqualifier dis, the pair is *eligible* when $[\![\mathrm{req}]\!](c) = \mathsf{T}$ and $[\![\mathrm{dis}]\!](c) \neq \mathsf{T}$, with the disqualifier evaluated after the requirement and sharing its binds.

A requirement passes only on T. A disqualifier blocks only on T. A disqualifier that evaluates to U, such as a request not to be contacted that has simply never been recorded, does not block a plan, since otherwise no plan could ever pass, but the unknown is recorded in the evaluation trace and appears in the output of the planning and explanation commands.

3.3 Monotonicity and Its Limits

Lemma 3.5 (Kleene connectives are monotone)**.** *The functions* min, max *and* $\neg$ *on* $\mathbf{K}_3$ *are monotone with respect to* $\sqsubseteq$*.*

Proof. Each function is determined by a finite table. If $x \sqsubseteq x'$ then either $x = x'$ or $x = \mathsf{U}$. In the second case $\min(x,y) \in \{\mathsf{U}, \min(x',y)\}$ because $\min(\mathsf{U},y) = \mathsf{F}$ only when $y = \mathsf{F}$, in which case $\min(x',y) = \mathsf{F}$ as well, and otherwise $\min(\mathsf{U},y) = \mathsf{U} \sqsubseteq \min(x',y)$. The argument for max is dual, and for $\neg$ it is immediate since $\neg\mathsf{U} = \mathsf{U}$. □

Information-monotonicity of the connectives is not sufficient for monotonicity of whole conditions with respect to the addition of evidence, because has can move from F to T when a positive record is added to a store that had only a negated one, and because missing and the binding of the latest record both move against the order. We therefore isolate a fragment on which the property holds.

Definition 3.6 (Positive fragment)**.** Let $\mathcal{L}_0$ be the set of expressions that contain no not, no $\mathsf{missing}$ and no bind names, and that use has, eq, in, cmp, all and any only. A *positive extension* of a store E is a store $E' \supseteq E$ in which neither E nor $E' \setminus E$ contains a negated record.

Proposition 3.7 (Monotonicity on the positive fragment)**.** *If $\varphi \in \mathcal{L}_0$ and E' is a positive extension of E, then $[\![\varphi]\!](E) \leq [\![\varphi]\!](E')$ in the truth order, for every fixed π, g and d^*.*

Proof. By induction on φ. For $\mathsf{has}(\varsigma)$ with no negated records the value lies in $\{\mathsf{U}, \mathsf{T}\}$ and equals T iff $M^+ \neq \emptyset$. Since $M^+(E) \subseteq M^+(E')$, the value cannot decrease. The forms eq, in and cmp do not depend on E because no bind name occurs. For all and any the claim follows from monotonicity of min and max in the truth order. □

Corollary 3.8 (Removal of the unique witness)**.** *Let* $\mathrm{req} = \mathsf{all}(\psi_1, \ldots, \psi_n) \in \mathcal{L}_0$ *with some* $\psi_i = \mathsf{has}(\varsigma)$*, and suppose a record e is the only member of $M^+(\varsigma)$ in a negation-free store E. Then in $E \setminus \{e\}$ the requirement is not* T*, so no plan depending on it is eligible.*

Proof. In $E \setminus \{e\}$ the set $M^+(\varsigma)$ is empty and no negated record exists, hence $[\![\psi_i]\!] = \mathsf{U}$ and the minimum is at most U. □

Remark 3.9 (Recency and the failure of monotonicity with binds)*.* The goal for sharing a revision binds the latest matching revision request and compares its date with the project's revision date. Adding a later request can therefore turn a true comparison into a false one. This is intended behavior and not a defect: a recipient whose request postdates the revision has already been served by that revision, and the corpus contains such a recipient, who abstains. The consequence for the specification is that conditions using binds must be reasoned about directly and not through Proposition 3.7.

4 Preference Resolution

Author preferences are layered. Global rules cover matters such as making exactly one concrete request and avoiding invented familiarity. Project rules can prefer a short title or prescribe how to refer to a version. Occasion rules set length and directness. Resolution is a function of the occasion o, the project j and a date d, and its order is fixed: hard constraints first, then occasion settings, then project preferences, then author defaults.

Definition 4.1 (Applicable preferences)**.** For (o, j, d) let $A(o, j, d)$ be the set of preferences whose scope is global, or is occasion:o, or is project:j, and whose window, when present, contains d. For a preference q let $\mathrm{lvl}(q) \in \{0, 1, 2\}$ be 0 for global, 1 for project and 2 for occasion scope, and let $\mathrm{rank}(q) = (\mathrm{lvl}(q), q.\mathrm{prio})$ under the lexicographic order.

Definition 4.2 (Resolution)**.** Keys fall into two classes. The *union keys* are the lists of forbidden characters and forbidden phrases. For a union key κ the resolved value is the union of the values of all applicable preferences with key κ. For every other key let A_κ be the applicable preferences with that key. If A_κ contains hard preferences, the resolved value is their common value, and resolution is a *conflict* if they disagree. Otherwise let W_κ be the preferences of maximal rank in A_κ, and the resolved value is their common value, or a conflict if they disagree.

Two properties of this definition deserve comment. A hard preference of any scope overrides a soft one of any scope, which is how a global prohibition survives an occasion setting that would otherwise take precedence. And two hard preferences that disagree are a conflict even when they have different scopes, because hardness expresses equal and absolute authority.

Theorem 4.3 (Well-definedness)**.** *Resolution is a total function from (o, j, d) to the set of key-value maps together with the value* conflict*. Its result does not depend on the order in which preferences are enumerated.*

Proof. The set $A(o, j, d)$ is determined by the scopes and windows alone. For each key the result is computed from the set A_κ by taking a union, a maximum under a total preorder, or the common value of a set, and each of these operations is invariant under reordering of its argument and total on finite sets, with disagreement mapped to the single value conflict. □

Proposition 4.4 (Scope of conflict validation)**.** *Validation resolves preferences for every pair $(o, j) \in (\mathcal{O} \cup \{\bot\}) \times (\Pi \cup \{\bot\})$ at the reference date and at each endpoint of each effective window, and reports any conflict found. It therefore certifies conflict-freedom at those finitely many points only.*

Proof. The statement describes the procedure. The restriction is necessary to state because conflict is not monotone in the set of applicable preferences: adding a higher-ranked soft preference can remove a conflict, so a conflict-free check at window endpoints does not imply conflict-freedom on the open intervals between them. □

The corpus declares no effective windows, in which case the set of checked points is complete.

Example 4.5. For the occasion `followup` and the project *Proof Abundance* the resolved maximum body length is 120 words, from an occasion-scoped preference that outranks the global default of 160. The resolved title style is `short`, from a project-scoped preference that outranks the global value `canonical`. The resolved request limit is one, from a hard global preference that no occasion or project setting can override.

5 Planning

5.1 Candidate Pairs

For a slot $s = (\mathrm{id}, \mathrm{mode}, r, o, j?, p?)$ the planner enumerates

$$C(s) = \{(g, \pi) \ : \ \mathrm{purpose}(g) \in \mathrm{allowed}_o,\ \pi \in \Pi_g,\ p? \in \{\bot, \mathrm{purpose}(g)\},\ j? \in \{\bot, \pi\}\}.$$

A pair is *active* when d^* lies in the goal's window. An inactive pair is recorded as such and is not evaluated. For each active pair the planner evaluates the requirement and the disqualifier in a fresh context and records, for each leaf evaluated, the operation, its value and a short description, so that every pass or fail can be reconstructed.

5.2 Scoring

Among eligible pairs, the planner ranks relevance to the recipient, continuity with an existing exchange, strength of the evidence relied on, and fit with the author's current priorities. The rule is an integer sum under a version tag, here score-v1.

Definition 5.1 (Score)**.** Let B be the set of evidence records bound during evaluation of the pair (g, π), and let E_{plan} be the plan-usable records of the recipient. Define

$$\rho = \begin{cases} 40 & \exists e \in B :\ e.j = \pi.\mathrm{id}, \\ 25 & \text{else if } \exists e \in E_{\mathsf{plan}} :\ e.p \in \mathcal{I},\ e.j = \pi.\mathrm{id}, \\ 0 & \text{otherwise}, \end{cases}$$

$$\kappa = \begin{cases} 20 & \exists e \in E_{\mathsf{plan}} :\ e.p = \mathrm{corresponded_with}, \\ 0 & \text{otherwise}, \end{cases}$$

where $\mathcal{I} = \{\mathrm{downloaded}, \mathrm{read}, \mathrm{commented_on}, \mathrm{requested_revision}\}$ is the set of interest predicates. Let w be the strength table of Table 1 and define

$$\sigma = \max_{e \in B} w(e.p) \ +\ 10 \cdot \mathbf{1}\big[\exists e \in B :\ 0 \le d^{*} - e.t \le 365\big], \qquad \mathrm{score}(g, \pi) = \rho + \kappa + \sigma + g.\mathrm{prio}.$$

Table 1: Evidence strength table w used by score-v1.

Predicate	w
commented_on	40
requested_revision	35
offered_to_help_with	30
read	20
downloaded	5
all others	0

5.3 Selection

Definition 5.2 (Selection)**.** Let $\mathcal{E}$ be the eligible pairs sorted by $(-\mathrm{score},\ g.\mathrm{id},\ \pi.\mathrm{id})$ and let $s_{\max}$ be the top score. For a band $\beta \ge 0$ let $\mathcal{E}_\beta = \{c \in \mathcal{E} : \mathrm{score}(c) \ge s_{\max} - \beta\}$, kept in sorted order. If $|\mathcal{E}_\beta| = 1$ the planner selects its member. Otherwise it selects $\mathcal{E}_\beta\big[h(\mathrm{seed}, s.\mathrm{id}, \mathrm{plan}, \mathrm{tie}) \bmod |\mathcal{E}_\beta|\big]$.

Randomness therefore operates only after eligibility and relevance have been settled, and with $\beta = 0$ only among exact ties. When a recipient with no stronger evidence qualifies for the typesetting goal on three older essays at equal score, the seed alone decides which essay is named, and the decision is reproducible.

5.4 Abstention

Definition 5.3 (Abstention reason)**.** If the recipient's language has no grammar rule, the slot abstains with `unsupported_language`. Otherwise, if $\mathcal{E} = \emptyset$, the reason is the first applicable of the following: `no_admissible_goal` if $C(s) = \emptyset$; `recipient_excluded` if some pair had a true requirement and a true disqualifier; `no_active_goal` if every pair was inactive; `no_supported_recipient_context` if needr_o holds; and `no_eligible_plan` otherwise.

Proposition 5.4 (Soundness of planning)**.** *If a slot yields a plan, then the selected pair satisfied* $[\![\text{req}]\!] = \mathsf{T}$ *and* $[\![\text{dis}]\!] \neq \mathsf{T}$ *in the frozen snapshot, and every leaf that contributed to that outcome is recorded in the trace. A plan is never produced from an unknown requirement.*

Proof. Selection draws only from $\mathcal{E}$, which consists of eligible pairs by Definition 3.4. The trace is appended at every leaf evaluation. □

Example 5.5 (Ranking)**.** A recipient has commented on page 18 of *Proof Abundance* on 2026-10-04 and offered help with LaTeX on the same project on 2026-10-07. With $d^* =$ 2026-11-11, the substantive follow-up pair has $\rho = 40$ because the bound comment concerns that project, $\kappa = 0$, $\sigma = 40 + 10 = 50$ and $g.\text{prio} = 20$, for a score of 110. Each of the three typesetting pairs has $\rho = 0$ because the offer concerns a different project from each essay and no interest record concerns any essay, $\kappa = 0$, $\sigma = 30 + 10 = 40$ and $g.\text{prio} = 10$, for a score of 50. The follow-up is selected. If the slot carries a purpose constraint naming typesetting, the three typesetting pairs are the only candidates, they tie at 50, and the seed selects among them.

5.5 The Plan Object

Definition 5.6 (Plan)**.** A plan is a tuple

$$P = (s, \text{lang}, o, \text{purpose}, g, \pi, \text{intent}, \text{open}, \text{core}, \\ \text{ask}, \text{close}, \text{asp}, \text{fld}, \mathcal{C}, \text{cons}, \text{score}).$$

Its components are the slot, the recipient's language, the occasion, the purpose, goal and project; the subject intent; the opening move, which is `cite_recipient_evidence` when some recipient-derived certificate may be mentioned and `introduce_task` otherwise; the core claim; the primary request ask = (action, obligation); the closing intent; the aspect partition $\text{asp} = (F, O, X)$; the field map fld; the set $\mathcal{C}$ of certificates; and the resolved constraint map cons. A plan also carries a plan signature and a content hash.

The obligation of every request in the present grammar is `ask_if_able`: the recipient is invited to help and is not instructed to. A grammar rule that expressed a demand would carry a different obligation and would be rejected at realization.

6 Certificates and the Assertion Calculus

6.1 Certificates

Each assertion that a draft may make carries a small certificate. The purpose of the certificate is to make the scope of a claim explicit, so that evidence collected for one use or one recipient cannot silently support another.

Definition 6.1 (Certificate)**.** A certificate is $\gamma = (\mathrm{id}, k, \mathrm{prop}, \mathrm{src}, \mathrm{rule}, \mathrm{scope}, u, \mathrm{fl})$ with a kind $k \in \mathcal{K}$, a proposition string, a list of source identifiers, the inference rule used (the value `direct` when none), a scope $(r, o, \mathrm{purpose}, \pi)$, an allowed use $u \in \{\mathsf{mention}, \mathsf{plan_only}\}$, and a field map fl of placeholder values the certificate contributes.

The kind set is $\mathcal{K} = \mathcal{K}_{\mathrm{rec}} \cup \mathcal{K}_{\mathrm{proj}}$ with

$$\mathcal{K}_{\mathrm{rec}} = \{\text{recipient_offered_help},\ \text{recipient_commented},\ \text{recipient_requested_revision},,\ \text{recipient_downloaded},\ \text{recipient_read_passage},\ \text{prior_exchange}\}$$

$$\mathcal{K}_{\mathrm{proj}} = \{\text{project_older_essay},\ \text{project_in_progress},\ \text{project_revision},\ \text{project_argument},\ \text{project_url}\}.$$

Pragmatic implications are treated exactly as factual assertions. The sentence "Thanks again for reviewing the draft" requires support both for the review and for the implied earlier acknowledgment, and the two are represented by the kinds recipient_commented and prior_exchange.

6.2 Construction

Direct certificates of recipient kinds are built from a fixed correspondence between kinds and predicates, shown in Table 2. For a kind k with predicate p and project scope ps, the source is the latest record (maximal in (t, id)) among positive, plan-usable records of that predicate for the recipient, restricted to the plan's project when ps is `project`.

Table 2: Certificate kinds derived directly from evidence.

Kind	Predicate	Project scope	Fields contributed
recipient_offered_help	offered_to_help_with	any	offer_topic
recipient_commented	commented_on	plan project	comment_topic, locator
recipient_requested_revision	requested_revision	plan project	none
recipient_downloaded	downloaded	plan project	none
prior_exchange	corresponded_with	any	none

The use of a direct certificate is mention if and only if the source record's allowed-use set contains mention, and plan_only otherwise. Inference rules then extend the set in a single step: for each rule (id, k, k') and each certificate of kind k that carries a locator whenever the rule requires one, a certificate of kind k' is added with the same sources and use. In the reference grammar the only rule is that a comment with a locator licenses the kind recipient_read_passage. Project certificates are derived from the project record and always have use mention: an older essay yields project_older_essay, status `revising` yields project_in_progress, the presence of a revision and revision date yields project_revision with the revision note as a field, a stated argument yields project_argument, and every project yields project_url.

Definition 6.2 (Mentionable kinds and fields)**.** For a plan P with certificate set $\mathcal{C}$ let $\mathcal{M}(P) = \{\gamma.k : \gamma \in \mathcal{C},\ \gamma.u = \mathsf{mention}\}$. The field map $\mathrm{fld}(P)$ consists of the author's name, the recipient's

given name, the project title in the resolved style, the canonical title, the URL, the aspect values argument and revision_note when the project has them, one defect kind chosen as pick(defect_kinds; seed, s.id, defect), and the union of the field maps of all certificates in $\mathcal{C}$ of use mention with defined values.

The last clause is the formal statement of plan-only isolation: values contributed by a certificate whose use is plan_only do not enter $\mathrm{fld}(P)$, so no template that uses them can be completed.

6.3 Aspect Relegation

Aspect relegation becomes an explicit planning operation. The goal's partition (F, O, X) of aspect names says which aspects of the project are foregrounded, which may appear if space permits, and which are excluded. For a typesetting request the title and the kind of defect are foregrounded, the URL is optional, and the formal argument and the revision note are excluded. For a substantive follow-up the argument is foregrounded and the defect kinds are excluded. Excluded aspects remain available to the planner without appearing in the email, and Section 7 states the check that enforces this on the finished text.

7 Surface Realization

7.1 Rules

Definition 7.1 (Grammar rule)**.** A grammar rule is a tuple

$$r = (\mathrm{id}, \mathrm{slot}, \mathrm{Pur}, \mathrm{lang}, \mathrm{Reg}, \mathrm{Dir}, \mathrm{asserts}, \mathrm{implies}, \mathrm{needs}, \\ \mathrm{intro}, \mathrm{follows}, \mathrm{move}, \mathrm{cons}, \mathrm{req}, \mathrm{act}, \mathrm{obl}, \mathrm{text}, \mathrm{can}).$$

The slot lies in the set $\{\mathrm{subject}, \mathrm{greeting}, \mathrm{reason}, \mathrm{context}, \mathrm{request}, \mathrm{support}, \mathrm{closing}, \mathrm{signoff}\}$. The purposes Pur form a set or the wildcard. The register and directness sets state the compatible registers and levels of directness. The fields asserts and implies are sets of certificate kinds, needs and intro are sets of discourse referents, follows is a set of slots that may directly precede the rule, move is a discourse move label, cons is a grammatical construction label, $\mathrm{req} \in \{0, 1\}$ counts requests, act and obl are present exactly when slot = request, text is a template, and can marks canonical rules.

Templates contain placeholders $\{x\}$ with x in a fixed vocabulary. Two maps derive requirements from the placeholders of a template. The map asp sends title placeholders to the aspect title, the URL to url, the defect to defect_kinds, the argument to argument and the revision note to revision_note. The map kd sends the offer topic to recipient_offered_help, the comment topic and the locator to recipient_commented, the revision note to project_revision, the argument to project_argument and the URL to project_url. The *need set* of a rule is

$$\mathrm{need}(r) = r.\mathrm{asserts} \cup r.\mathrm{implies} \cup \mathrm{kd}(\mathrm{ph}(r)),$$

so a rule that uses a comment topic cannot be used unless a mentionable comment certificate exists, whether or not its author declared the dependency.

7.2 Eligibility

Definition 7.2 (Rule eligibility)**.** A rule r is eligible for a plan P when all of the following hold: its purposes contain the plan's purpose or the wildcard; its language equals the plan's language; its register set meets the resolved register set; its directness set meets the resolved directness set; need$(r) \subseteq \mathcal{M}(P)$; ph$(r) \subseteq$ dom(fld(P)); asp(ph(r)) $\subseteq F \cup O$ and asp(ph(r)) $\cap X = \emptyset$; and, for request rules, r.act equals the plan's action.

7.3 Discourse Orders and Candidate Identifiers

The separation of planning from realization follows the standard architecture of natural language generation systems [2]. A draft is realized in one of three discourse orders, each a sequence of slots.

$$\omega_1 = (\text{greeting, reason, context, request, support, closing, signoff}),$$
$$\omega_2 = (\text{greeting, request, reason, support, closing, signoff}),$$
$$\omega_3 = (\text{greeting, context, reason, request, support, closing, signoff}).$$

These capture the choice to lead with the request, to establish why the recipient was chosen first, or to introduce the project first. The slot support is optional and every other slot is required. The subject line is realized by a subject rule using the same plan and the same checks as the body.

Definition 7.3 (Candidate)**.** A candidate for P is a tuple $c = (\omega, r_{\text{sub}}, r_1, \ldots, r_n)$ with $\omega \in \{\omega_1, \omega_2, \omega_3\}$, a subject rule, and one rule (or the symbol $-$ for the optional slot) per slot of ω. Its identifier is the string $\omega \,|\, r_{\text{sub}}.\text{id} \,|\, r_1.\text{id} \,|\, \cdots$. The set of all candidates for P whose rules are eligible is $\Omega(P)$, and

$$|\Omega(P)| = \sum_{\omega} |R_{\text{sub}}| \prod_{x \in \omega} \left(|R_x| + \mathbf{1}[x = \text{support}]\right)$$

where R_x is the set of eligible rules for slot x.

A candidate is built from its identifier alone. Any draft can therefore be reconstructed from a plan and a rule path, which is what lets an awkward sentence be traced to its construction and turned into a focused regression case.

7.4 Validity

Text is assembled by filling placeholders from fld(P). A greeting forms its own paragraph and a sign-off forms the last. The request and the closing each begin a new paragraph, and all other slots join the current paragraph.

Definition 7.4 (Validity)**.** A candidate c is *valid* for P, written Valid(P, c), when all of the following conditions hold. Let cons $= P$.cons.

V1 The identifier is well formed and has one entry per slot of its order.

V2 Every rule named is eligible for P and belongs to the slot in which it appears.

V3 For every rule with a non-empty follows set, the slot immediately preceding it in the draft belongs to that set.

V4 For every rule, r.needs is contained in the union of the intro sets of the rules before it.

V5 No discourse move occurs twice, unless the rules declare it repeatable.

V6 The request rule has action and obligation equal to P.ask.

V7 The request counts sum to exactly one and do not exceed cons(max_requests).

V8 No placeholder remains unresolved in the subject or body.

V9 For every trigger (pattern, k) of the lexicon that matches the subject or body, $k \in \mathcal{M}(P)$.

V10 No forbidden character, forbidden phrase or emoji occurs.

V11 The body has at most cons(max_words) words.

V12 The subject has at most 90 characters.

V13 The resolved title occurs in the body at most cons(title_max_mentions) times.

V14 For every excluded aspect $x \in X$, no value of that aspect occurs, compared case-insensitively, in the subject or body.

V15 At most one rule of the draft uses the URL aspect.

7.5 Properties

Theorem 7.5 (Certificate coverage)**.** *If* Valid(P, c)*, then every rule of* c *satisfies* need$(r) \subseteq \mathcal{M}(P)$*, and every trigger of the lexicon that matches the rendered subject or body has its kind in* $\mathcal{M}(P)$*. Consequently each explicit assertion, each declared pragmatic implication and each lexically detectable pragmatic implication of the draft is covered by a certificate of use* mention*.*

Proof. The first claim follows from V2 and Definition 7.2. The second is V9. The consequence restates these two facts using Definition 6.2. □

The theorem is relative to what rules declare and what the lexicon can detect. A lexical trigger is a heuristic safeguard against mislabelled rules. It cannot exhaust the pragmatic implications of free prose, and Section 14 returns to this.

Proposition 7.6 (Request preservation)**.** *Let* c *be valid for* P*. Then* c *contains exactly one request rule, and its action and obligation equal those of* P*. Hence no choice among direct questions, conditional invitations and statement-question forms can change the requested action or its obligation.*

Proof. V7 gives exactly one request and V6 equates its action and obligation with the plan's. Definition 7.2 already restricts request rules to the plan's action. □

Proposition 7.7 (Plan-only isolation)**.** *Suppose that for a kind* k *every certificate of kind* k *has use* plan_only*. Then no valid candidate contains a rule with* $k \in$ need(r)*, and no valid candidate fills a placeholder whose only source is a certificate of kind* k*.*

Proof. Then $k \notin \mathcal{M}(P)$, so rules with $k \in \mathrm{need}(r)$ are ineligible. The values of such certificates are excluded from $\mathrm{fld}(P)$ by Definition 6.2, so every rule using them has $\mathrm{ph}(r) \not\subseteq \mathrm{dom}(\mathrm{fld}(P))$ and is ineligible. □

Proposition 7.8 (Aspect exclusion)**.** *For every valid candidate and every $x \in X$, neither any rule used nor the finished text contains the value of aspect x.*

Proof. Rule-level exclusion is Definition 7.2. Text-level exclusion is V14. □

Remark 7.9 (Referents)*.* Condition V4 makes pronoun resolution a checked property. A request phrased as "look through it for overfull lines" needs the project to have been introduced earlier in the draft, so it cannot occur first in a request-leading order, where only the full-title variant of the request is admissible. Validation additionally rejects any rule whose template contains a pronoun or definite reference but declares no referent and contains no title placeholder.

7.6 Candidate Generation and Fallback

For each eligible plan the renderer samples a bounded set of candidate identifiers. For $i = 0, \ldots, K-1$ the order is $\mathrm{pick}(\{\omega_1, \omega_2, \omega_3\}; \mathrm{seed}, s.\mathrm{id}, \mathrm{render}, \mathrm{order}, i)$, and each slot rule is drawn likewise with the slot name in place of the stage component, with $-$ included among the options for the support slot. The sampled identifiers are deduplicated and sorted. Each is built and tested for validity, and the valid ones form the set $V(P)$.

Sampling does not depend on validity. This is a deliberate property used in Section 8: the set of sampled identifiers is a function of the plan, the seed and the eligible rule lists alone. If $V(P) = \emptyset$, the renderer tries a *canonical rendering*, the order ω_1 with the first canonical eligible rule in each required slot and no support. If that candidate is invalid the slot records a rendering failure whose reason names the first violated condition. The canonical rendering is an escape from gaps in the sampled set and does not widen what the grammar can say.

8 Determinism and Reproducibility

8.1 Seeds and Selection

Every random choice in the system is a value of h over the master seed, the slot identifier, a stage name and a choice identifier. The reference date supplies every date comparison. The planner and the renderer are therefore pure functions: for a fixed snapshot, engine, seed, date, band, sample size and slot, they return the same plan and the same candidate set on every machine and in every process.

Definition 8.1 (Selection rank)**.** For a slot s, a seed and a candidate identifier c, let $\mathrm{rank}_s(c) = (h(\mathrm{seed}, s.\mathrm{id}, \mathrm{select}, c),\ c)$ under the lexicographic order. The *selected* candidate of a non-empty set V is $\arg\min_{c \in V} \mathrm{rank}_s(c)$, and the order of fallbacks under a distinctness constraint is increasing rank.

Rank is a function of the candidate identifier and the slot alone. It does not depend on the other members of V, and that is what makes the next result possible.

Theorem 8.2 (Stability of selection)**.** *Let P and P' be plans for the same slot with the same eligible rule lists, so that the sampled identifier sets coincide, and suppose every valid candidate of P' is valid for P, that is $V(P') \subseteq V(P)$. Let c be the candidate selected from $V(P)$. If $c \in V(P')$ then c is also selected from $V(P')$. If $c \notin V(P')$ then the selection changes, to the minimum-rank member of $V(P')$ or to a failure if $V(P') = \emptyset$.*

Proof. Since rank_s is a fixed total order on identifiers, the minimum of a subset that contains the minimum of the superset is that same element. If the minimum of $V(P)$ was removed, the minimum of the remaining set is a different element, or the set is empty. □

Corollary 8.3 (Predictable effect of tightening a constraint)**.** *Let a change to the preferences affect only the conditions that read the resolved constraint map, namely V7, V10, V11 and V13, and not the resolved register or directness, so that eligible rule lists are unchanged, and let it only tighten those conditions. Then the draft of a slot changes if and only if the previously selected candidate violates the new constraint.*

This is the form in which the final criterion of the specification is met. For example, lowering the permitted number of title mentions for one occasion changes exactly those drafts whose chosen text mentioned the title more often than the new limit. Section 12 describes the test that checks this over a batch. Because rank is computed per candidate and not by position in a list of valid candidates, tightening a constraint cannot reshuffle drafts it does not invalidate.

8.2 Execution Independence

Batch generation has two stages.

Definition 8.4 (Stages)**.** *Stage A* computes, for each slot, the plan or abstention and the bounded candidate list, and commits them to a staging table. It is a pure function of the slot. *Stage B* traverses the slots in increasing index. For each slot it appends ledger events, and for planned slots it selects the first candidate in increasing rank that is not already reserved (when distinctness is required), reserves its fingerprint, re-renders the draft from the plan and the candidate identifier, and commits the draft, the state change and the events in one transaction group.

Theorem 8.5 (Independence from schedule)**.** *Fix a manifest, a snapshot and an engine identity. The final contents of the draft table, the fingerprint reservations and the ledger are the same for every number of workers, every partition of slots into stage A chunks, and every pattern of interruption and resumption in which each chunk and each transaction group commits atomically.*

Proof. Stage A results are pure functions of the slot, so their values do not depend on which worker computed them or on the order in which chunks complete, and a chunk that was committed before an interruption is not recomputed to a different value. Stage B is a single sequential process over slots in index order. The state after processing slot i, namely the reserved fingerprints, the ledger head and the table contents, is determined by the state after slot $i - 1$ and the stage A result for slot i. A transaction group either commits fully or not at all, so after an interruption the committed state equals the state after some prefix of the slots, and resumption proceeds from the first slot not in the prefix. By induction on i the final state equals the state of an uninterrupted serial run. □

Two points are essential to the proof and to the implementation. Ledger events for planning, abstention, rendering and validation are written in stage B and not in stage A, since stage A chunks may complete in any order and would otherwise make the ledger depend on scheduling. And the uniqueness constraint on the pair of batch and slot prevents a retried slot from producing a second result.

8.3 Engine Identity on Resume

The preceding theorem assumes the same engine throughout. A resumed run that processed pending slots with changed code while retaining results of the old code would break reproducibility, and the check that the manifest and snapshot match is not enough to prevent this.

Definition 8.6 (Engine identity)**.** The engine identity is the triple $(\text{version}, \text{schema}, \text{code})$ where version and schema are the engine and schema version numbers and $\text{code} = \mathrm{H}(\text{canon}(\{(\text{name}, \mathrm{H}(\text{bytes})) : \text{ engine source files}\}))$. The identity is recorded when a batch is created.

Proposition 8.7 (Resume check)**.** *A run directory is resumed only if its recorded manifest hash, snapshot hash and engine identity equal the current ones. If any component of the identity differs, resumption stops before any pending slot is processed and names the differing components. A directory with no recorded identity cannot be resumed.*

Proof. The procedure compares the three recorded values with the current ones in turn and raises an error on any mismatch, before the first stage A chunk is scheduled. The source hash is included so that a code change without a version change is detected as well. □

The comparison of source digests is conservative. A harmless edit to a comment changes the digest and forces a restart. The cost of a spurious restart is bounded by the cost of the batch, whereas the cost of a missed difference is a silently inconsistent batch.

9 Diversity, Capacity and Distinctness

9.1 Three Separate Measures

A single count of distinct drafts conflates variation of three kinds that have different meanings. A draft can differ from another only because a recipient's name was substituted, or only because a synonymous clause was chosen, or because it has a different purpose, evidence base and request. The program therefore records three measures and reports them separately.

Definition 9.1 (Normalization and fingerprint)**.** Let $\nu(x)$ be the string obtained from x by Unicode normalization to NFC, deletion of every substring that matches the address pattern, collapse of every maximal run of whitespace to a single space, and trimming. The *text fingerprint* of a draft with subject a and body b is the first 128 bits of $\mathrm{H}(\nu(a) \,\|\, \text{newline} \,\|\, \nu(b))$.

Definition 9.2 (Structural and plan signatures)**.** The *structural signature* of a candidate is the first 48 bits of $\mathrm{H}(\omega \,\|\, \text{“:”} \,\|\, \text{cons}(r_{\text{sub}}).\text{cons}(r_1).\cdots.\text{cons}(r_n))$, which depends on the discourse order and the sequence of grammatical constructions and not on the words chosen or the names

used. The *plan signature* of a plan is the first 48 bits of H applied to the concatenation of its purpose, project, requested action, opening move, the sorted recipient-derived mentionable kinds, the sorted foregrounded aspects and the chosen defect kind. It measures diversity of purposes, evidence categories, requests and foregrounded material.

A change of name alters the text fingerprint and leaves both other signatures unchanged, so name substitution contributes to the first measure only. The audit report prints the number of distinct values of each measure.

Proposition 9.3 (Fingerprint collisions)**.** *Under the usual idealization of SHA-256 as a random function, the probability that two distinct normalized texts among n share a fingerprint is at most $n(n-1)/2^{129}$. For $n = 10^6$ this is less than 1.5×10^{-27}.*

Proof. This is the standard birthday bound on 128-bit values: there are $\binom{n}{2}$ pairs and each collides with probability 2^{-128}. The numeric bound follows from $\binom{10^6}{2} < 5 \times 10^{11}$ and $2^{128} > 3.4 \times 10^{38}$. □

9.2 Distinctness Constraints and Capacity

In experiment mode a batch may require distinct normalized text. Stage B then reserves each accepted fingerprint in a table keyed by batch and fingerprint, and a slot takes the first candidate in increasing rank whose fingerprint reservation succeeds. A slot for which every valid candidate is already reserved records the failure `variation_exhausted`.

Definition 9.4 (Capacity)**.** For a plan P the *capacity* $N(P)$ is the number of distinct normalized texts over all valid candidates, $N(P) = \left|\{F(c) : c \in \Omega(P),\ \mathrm{Valid}(P, c)\}\right|$ with F the text fingerprint. A scenario's capacity is the number of distinct texts over the plans it can produce, and is at least the largest capacity of any one of them.

Because every candidate is rendered by a deterministic function of its identifier, $N(P) \le |\Omega(P)|$, and equality fails only when distinct rule paths yield equal text or when candidates are invalid. The raw size $|\Omega(P)|$ is exact from Definition 7.3. The fraction of valid candidates was estimated by sampling and is reported in Section 13.

A requested batch size is a capacity target, not a promise. Two distinct mechanisms can leave a slot without a draft under a distinctness constraint, and they must not be confused. The first is exhaustion of the grammar, when the number of slots drawn from one scenario approaches its capacity. The second is the bounded sample: each slot examines only the K candidates it sampled, of which only v are valid, and a slot with small v fails whenever all of those few happen to be reserved, even when most of the grammar is untouched. Under an idealization in which the v valid candidates are independent uniform draws from a capacity of N texts and m texts are already reserved, the failure probability of a slot is approximately $(m/N)^v$, and the expected number of failures in a scenario with T trials and valid-count distribution $\Pr[v]$ is

$$\mathbb{E}[\text{failures}] \;\approx\; \sum_{m=0}^{T-1} \sum_{v \ge 1} \Pr[v]\,(m/N)^v.$$

The idealization ignores deduplication within a sample and the non-uniform distribution of texts under uniform sampling of rule paths, and Section 13 shows that it underestimates the

observed count. We use it only to indicate order of magnitude and to explain why increasing K helps, since each added valid candidate multiplies the failure probability by a factor of $m/N < 1$.

10 Ledger, Storage and Audit

10.1 Slot States and Events

Each slot of a batch occupies exactly one state at a time. The transitions are

$$\text{pending} \to \text{staged} \to \{\text{drafted},\ \text{abstained},\ \text{failed}\}, \qquad \text{drafted} \to \text{exported},$$

and no other transition is permitted. The first transition is made by stage A, the second by stage B, and the last by export. The ledger records events of the kinds shown in Table 3.

Table 3: Ledger event kinds and the transitions that write them.

Event	Written by	Meaning
batch_created	start	Manifest hash, snapshot hash, engine identity, seed, date, slot count
slot_planned	stage B	Plan hash, purpose, goal, project, score
slot_abstained	stage B	Abstention reason
slot_failed	stage B	Rendering failure or exhausted variation
slot_rendered	stage B	Fingerprint and rule path of the chosen candidate
slot_validated	stage B	Count of mentionable certificates backing the draft
slot_exported	export	Format and digest of the exported record
batch_completed	end	Counts by state

10.2 Hash Chain

Definition 10.1 (Ledger)**.** A ledger is a sequence of events $e_1, e_2, \ldots$ with $e_n = (\text{kind}, \text{slot}, \text{record}, \text{payload})$, together with digests defined by $\eta_0 = 0^{256}$ and $\eta_n = \mathrm{H}(\eta_{n-1} \,\|\, \text{canon}(e_n))$. The *record* field is the digest of the object the event concerns, such as a plan, a fingerprint or an exported line, linking the event to that object. The *head* is the last digest.

The construction is the standard hash chain [3], and it is deliberately free of wall-clock timestamps, so that two runs that agree on every decision produce the same head.

Theorem 10.2 (Tamper evidence)**.** *Assume* H *is collision resistant. If a stored ledger passes verification, that is, recomputing η_n from η_{n-1} and e_n reproduces the stored digest for every n, then it equals the ledger that produced its head, up to computationally infeasible collision search. Any modification, reordering or deletion of an event other than a suffix changes the head.*

Proof. Suppose two distinct sequences of events yield the same head. Take the last index at which the pair (η_{n-1}, e_n) differs between them. The two inputs to H at that index differ, while the outputs agree, which is a collision. A deleted or inserted event shifts the indices from that point, so the same argument applies at the first affected position. □

Two caveats limit the practical statement. Database triggers that reject updates and deletes make accidental modification fail, but they are not a cryptographic protection, because a party with write access to the file can remove them. The chain detects such edits afterwards, but truncation of a suffix is detected only if the head is recorded elsewhere. The audit therefore prints the head so that it can be stored independently.

10.3 Accounting

Definition 10.3 (Accounting identity)**.** Let $n_{\text{pend}}, n_{\text{stag}}, n_{\text{draft}}, n_{\text{abst}}, n_{\text{fail}}, n_{\text{exp}}$ be the numbers of slots in each state and N_s the number of slots in the batch. A batch is *complete* when $n_{\text{pend}} = n_{\text{stag}} = 0$, and then it must satisfy

$$N_s = n_{\text{draft}} + n_{\text{exp}} + n_{\text{abst}} + n_{\text{fail}}.$$

The ledger *agrees* with the tables when the number of abstention events equals n_{abst}, the number of failure events equals n_{fail}, the number of validation events equals $n_{\text{draft}} + n_{\text{exp}}$, and the number of distinct slots with an export event equals n_{exp}.

Proposition 10.4 (Maintenance of the identities)**.** *After every committed transaction group of stage B and every committed export, the ledger agrees with the tables, and the accounting identity holds whenever the batch is complete.*

Proof. Each slot transition writes its ledger events and its state change in the same transaction. A slot ends in exactly one terminal state through exactly one of the abstention, failure or validation paths, and moves from drafted to exported exactly once because the export event is written only for slots found in the drafted state. Atomic commit [5] makes the observable state equal the state after some whole number of such transitions. □

The audit verifies the chain, the agreement of the ledger with the tables, and the accounting identity, and exits with a failure status when any of them is violated. A completed run that produced only partial output therefore remains visible as such, and failures are reported by reason and not folded into totals.

10.4 Storage Layout

Table 4 lists the batch tables. Two choices are made for scale. Stage A stores the plan as a compressed blob and each sampled candidate as an identifier, a fingerprint and a structural signature, with no text, because the text is a deterministic function of the plan and the identifier and is re-rendered at selection. And stage B checks, after re-rendering, that the fingerprint equals the stored one, and records a failure `rerender_mismatch` if it does not, so a grammar change between stages cannot pass unnoticed even apart from the engine identity check.

Table 4: Tables of the batch database.

Table	Content
meta	Manifest and its hash, snapshot path and hash, engine identity, slot count, timing, completion flag
slot_state	One row per slot with index, state and reason; unique on batch and slot
stage_a	Stage A result per slot: status, reason, compressed plan, candidate list, purpose, plan signature
draft	Chosen subject and body, fingerprint, structural and plan signatures, rule path, purpose, project
fingerprint	Reserved fingerprints, unique on batch and fingerprint
ledger	Hash-chained events; updates and deletes refused by trigger

10.5 Export

Export reads committed drafts in stable slot order and writes each output through a temporary file followed by an atomic rename. Formats are JSON Lines for analysis, plain text for review, and `.eml` drafts marked unsent for a mail client. Recipient addresses are resolved only for the `.eml` format, from a separate file keyed by address reference, and an unresolved reference yields an explicit placeholder in the header. Experiment batches refuse `.eml` export, since their recipients are fictional. The JSON Lines export can omit evidence propositions, which is the form intended for audit, where recipients appear only by stable identifier.

11 Offline Maintenance

The generator runs with no model. A local model served by Ollama, such as Granite, is an optional tool for maintaining the rules, not part of generation.

Definition 11.1 (Maintenance packet)**.** A maintenance packet for a completed batch consists of anonymized failure and abstention traces, grammar coverage (the rules never used and the most used), the most repeated adjacent rule pairs with their shares of all drafts, the output of the test suite when requested, the sources of the grammar and of the evaluator, and a fixed instruction asking for a patch tied to observed examples that leaves factual certificates unchanged.

Anonymization maps each slot identifier x to a tag formed from a prefix of $\mathrm{H}(\text{maintenance} \,\|\, x)$ and carries no recipient names, so a packet reveals the structure of failures and nothing of the recipients. A model's reply, when requested, is written to a file and never applied.

Definition 11.2 (Repair cycle)**.** A repair begins with a precise observation o, such as the observation that direct requests following a long context paragraph often repeat the title. The maintenance pass identifies the discourse paths that produce the observation and proposes a patch Δ that turns the grammar G into G'. A regression fixture φ_o is a test that fails under G

and passes under G'. A patch is *accepted* when φ_o passes under G', the memory validates, the fixed regression corpus and the semantic tests pass under G', and the certificates of every plan in the corpus are unchanged.

The last condition is automatic for grammar patches, because certificates are produced by the planner from evidence and are independent of the grammar, but it is stated because a patch to the evaluator could change them. After acceptance the repair persists in the versioned inputs, and later runs receive its benefit without consulting any model. The title-limit test of Section 12 is an instance: it checks that the title occurs at most once in the occasion that limits it, that the request remains intact, and that the factual certificates are the same.

12 Acceptance Criteria

Acceptance begins with semantic tests. Each criterion below is stated as a formal property and is paired with the tests of the reference implementation that check it.

Table 5: Acceptance criteria, their formal statements and the checking tests.

	Criterion	Checking tests (names abbreviated)
A1	Removing the unique witness of a required record makes the dependent plan ineligible (Corollary 3.8)	removing_required_evidence
A2	Changing a fact outside the scope of a plan leaves plan and candidates identical	irrelevant_change
A3	Surface variation preserves the requested action and every asserted proposition (Theorem 7.5, Proposition 7.6)	variation_preserves_action; every_draft_traces
A4	Wording that implies a prior exchange fails without support	unsupported_prior_exchange; prior_exchange_requires_evidence
A5	Expired goals, conflicting preferences and unknown facts have their specified outcomes	expired_goal; revision_date_boundary; already_received; both preference-conflict tests; unknown_and_plan_only
A6	Plan-only evidence never reaches text (Proposition 7.7)	unknown_and_plan_only; downloaded_supports_only_downloaded
A7	Excluded aspects stay out of text (Proposition 7.8)	excluded_aspects
A8	Output is independent of workers and interruption, and a changed engine blocks resume (Theorem 8.5, Proposition 8.7)	identical_across_workers; interruption; resume_refuses_changed_engine; resume_no_duplicates
A9	Tightening a constraint changes exactly the predicted drafts (Corollary 8.3)	preference_change_predicts_drafts
A10	Ledger immutability, snapshot tamper detection, regeneration agreement and export accounting	ledger_append_only; snapshot_detects_edits; regeneration_matches; export_accounting; correspondence_mode_eml

The initial scale target is 10,000 synthetic drafts, with a review sample drawn across purposes, grammar rules and failure categories. A separate million-slot benchmark is intended to measure throughput, memory use, collision handling and restart behavior, and to compare output hashes across serial, parallel and interrupted runs. The more consequential test, in the words of the specification, is whether a project fact or a preference can be changed, the affected drafts predicted, and the engine verified to change those drafts and no others for the stated reason. Criterion A9 is the formal version of that test.

13 Reference Implementation and Measurements

13.1 Structure

The reference implementation is a Bash entry point over a Python engine of roughly two thousand lines, with SQLite for storage and TOML for the grammar, the preferences and the batch manifests. Table 6 maps the modules to the sections of this document.

Table 6: Modules of the reference implementation and the sections that specify them.

Module	Responsibility	Section
`memory`	Loading, validation and evaluation of evidence	2, 3
`prefs`	Layered preference resolution	4
`planner`	Eligibility, scoring, selection, certificates	5, 6
`rules`, `realize`	Grammar, candidates, validity, fallback	7
`batch`	Snapshots, stages A and B, audit, export	8, 10
`store`	SQLite schema, ledger and triggers	10
`maintain`	Maintenance packet	11

13.2 Corpus

The first release uses a fictional corpus, so that no real person appears in any experiment. It contains 15 evidence records, 3 goals, 5 projects, 12 recipients, 17 preferences and 9 trigger-lexicon entries. The grammar has 90 rules across three purposes, and the planner's canonical fallback has 15 rules. Appendix A gives the inventory.

13.3 Measurements

All figures below were measured on two cores of a development container and are single observations, not distributions.

Table 7: Measured batches. Abstentions in the experiment batches are by design.

Batch	Slots	Drafts	Abstained	Failed	Distinct texts
Experiment	10,008	5,838	4,170	0	5,838
Experiment, large	100,008	58,336	41,670	2	58,336

For the 10,008-slot batch the distinct counts were 76 structural signatures and 27 plan signatures. The 4,170 abstentions correspond to five recipients for whom no purpose has a supported reason, and are the expected outcome of the abstention rules of Section 5. The 100,008-slot batch took about 60 seconds, a rate near 1.6×10^3 slots per second. Both of its failures were `variation_exhausted`, and inspection showed that the two slots had one and two valid candidates among their twelve sampled, so by the analysis of Section 9 they reflect the sample bound and not the capacity of the grammar. The serial, two-worker and

interrupted-and-resumed runs of the 10,008-slot batch produced identical output hashes and ledger heads, which is an empirical instance of Theorem 8.5.

Capacity, estimated by sampling the path space, is about 6.0×10^5 to 7.8×10^5 distinct texts per scenario for most recipients, about 1.65×10^5 for the revision purpose, and about 1.08×10^5 for the comment case without mentionable evidence. These are estimates from sampling, not exact counts, and no confidence interval is reported.

13.4 Million-Slot Benchmark

A benchmark manifest defines a batch of one million slots, but it has *not been executed*. Extrapolating the measured rate gives about ten minutes and about 3.3 GB of storage, and these are extrapolations, not results. With 83,334 trials per scenario the batch approaches the capacity of the two smallest scenarios, so some `variation_exhausted` slots are predicted for them, and the report is designed to state the shortfall. The predicted counts rest on the idealization of Section 9, which underestimated the observed failures, and they should be treated as indicative only.

14 Limitations and Open Problems

This section states what the specification does not guarantee, so that the guarantees above are not read more broadly than they are given.

L1. Lexical incompleteness of the implication lint. Condition V13 detects implications by a finite trigger lexicon. Theorem 7.5 is therefore a guarantee relative to the lexicon: a phrase that implies an unsupported claim but matches no entry passes. The lexicon is extended by the repair cycle, which is reactive.

L2. Unknown disqualifiers. A disqualifier whose truth value is unknown does not block a plan, and the unknown is recorded. This favors availability over caution. A deployment that prefers caution can treat unknown as true for selected disqualifiers, at the price of more abstentions.

L3. Recency of binds. Binds take the most recent matching record, so adding evidence can change a bound variable and thereby change a downstream result (Remark 3.9). The monotonicity result is confined to the positive fragment.

L4. Partial preference validation. Validation checks the reference date and the window endpoints only. Preference sets that conflict on some other date in an overlapping window are detected at planning time, as an error for the affected plan, not at load time.

L5. Bounded sampling. Each slot examines K sampled candidates, so a distinctness constraint can fail a slot while unexamined valid candidates remain. The failure model of Section 9 is an idealization that underestimates.

L6. Fictional corpus. The measurements concern fictional recipients and fictional evidence. They show that the machinery is consistent and deterministic, and they say nothing about the quality of drafts for real correspondence, which requires review by the person on whose behalf drafts are written.

L7. Name substitution. A name change alters the text fingerprint and no other signature, so counts of distinct text overstate substantive variety. The three measures of Section 9 are reported separately for this reason.

L8. Language coverage. The grammar realizes English only, and plans for other languages abstain with the reason `unsupported_language`.

L9. Ledger scope. The hash chain gives tamper evidence, not tamper resistance, and detects suffix truncation only against an independently stored head.

L10. Unexecuted benchmark. The million-slot behavior is predicted, not measured.

A Grammar Rule Schema and Inventory

A.1 Rule Schema

A rule is a record with the fields of Table 8. The template is a string with placeholders in braces, from which the set of knowledge kinds the rule needs is derived (Section 7).

Table 8: Fields of a grammar rule.

Field	Meaning
id	Unique identifier within a slot and purpose
slot	One of subject, greeting, reason, context, request, support, closing, signoff
purpose	The purposes for which the rule is eligible
asserts	Propositions the rule's text commits to
implies	Propositions the rule's text suggests without stating
register	Formal, neutral or familiar, matched to the resolved register
directness	Direct, moderate or indirect, matched to the resolved directness
construction	A label naming the syntactic frame, used for the structural signature
template	Text with placeholders

A.2 Inventory

Table 9 lists the number of rules per slot for each of the three purposes, as reported by the reference implementation. Entries separated by slashes are the counts for the three purposes in the order of Section 5; single entries apply to all three, and the closing slot also has one rule specific to each purpose.

Table 9: Grammar rule counts by slot.

Slot	Rules per purpose
subject	5 / 5 / 4
greeting	4
reason	7 / 9 / 5
context	4 / 4 / 3
request	6 / 8 / 5
support	3 / 2 / 2
closing	6 shared + 1 per purpose
signoff	5

B Algorithms

Algorithm 1: Plan generation for one slot

Input: recipient a, date d, snapshot $\mathbf{K}_3$, resolved preferences
Output: a plan or an abstention with a reason

if *the recipient's language is unsupported* **then return** abstain(`unsupported_language`);
if *a is excluded by a hard preference* **then return** abstain(`recipient_excluded`);
$\mathcal{G} \leftarrow$ goals active at d;
if $\mathcal{G} = \emptyset$ **then return** abstain(`no_active_goal`);
$\mathcal{C} \leftarrow \{(g, p) \in \mathcal{G} \times \text{projects} : \text{Elig}(a, g, p)\}$;
if $\mathcal{C} = \emptyset$ **then return** abstain(`no_eligible_plan`);
choose $(g, p) \in \mathcal{C}$ by $(-\text{score}, g, p)$ with band and seeded tie-break;
build certificates, partition aspects, and attach the requested action;
return the plan;

Algorithm 2: Selection of a draft for one slot

Input: plan P, seed σ, slot i, batch fingerprint table
Output: a draft or a recorded failure

$S \leftarrow$ sample K candidate identifiers by σ and i;
$V \leftarrow \{c \in S : \text{Valid}(P, c)\}$;
if $V = \emptyset$ **then**
    render the canonical fallback; **if** *the fallback is invalid* **then return** failure;
foreach $c \in V$ *in increasing* $(h(\sigma, i, \text{“select”}, \text{cid}(c)), \text{cid}(c))$ **do**
    re-render c and check the stored fingerprint;
    if *no distinctness constraint or the reservation of its fingerprint succeeds* **then return** the draft;
return failure(`variation_exhausted`);

C Command-Line Interface

Table 10 summarizes the commands. Every command that reads a batch refuses to proceed on a mismatch of manifest hash, snapshot hash or engine identity, unless `-restart` is given to discard the partial run.

Table 10: Commands of the reference implementation.

Command	Effect
snapshot	Freeze the evidence into a hashed snapshot
validate	Check that the memory and the grammar are well formed
plan	Show the plan or the abstention for one recipient and date
generate	Run a batch from a manifest, resumable and parallel
audit	Verify the chain, the accounting identity and ledger agreement
explain	Show the plan, certificates and rule path of one slot
export	Write JSON Lines, text or `.eml` drafts
maintain	Write the offline maintenance packet

References

[1] S. C. Kleene. *Introduction to Metamathematics*. North-Holland, Amsterdam, 1952.

[2] E. Reiter and R. Dale. *Building Natural Language Generation Systems*. Cambridge University Press, 2000.

[3] R. C. Merkle. A certified digital signature. In *Advances in Cryptology: CRYPTO '89*, Lecture Notes in Computer Science 435, pages 218–238. Springer, 1990.

[4] National Institute of Standards and Technology. *Secure Hash Standard (SHS)*. FIPS PUB 180-4, 2015.

[5] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, 1993.