TraceGen: Traceable Grammar-Guided Controlled Realisation from Semantic Frames

Anonymous ACL submission

Abstract

Generated text is often used to test, augment, and analyse natural language processing (NLP) systems. However, in controlled-generation settings, a generated sentence is useful only if it preserves the intended meaning while realising the requested wording controls. We study this setting as controlled realisation from semantic frames, which specify an event and its participants, paired with wording controls such as modality, voice, and valency. To address this, we introduce TraceGen, a grammar-guided English realiser that returns both a sentence and a trace of the grammatical choices that generated it. We compare TraceGen with sentence-pattern templates and LLM baselines on 160 frame–control plans across five seeds. TraceGen obtains the highest exact-configuration recovery under the frame/control contract, 91.9%, with a strict denominator that includes explicit coverage failures. A verification audit then separates the result into complementary quality dimensions: templates are faithful but rigid, LLM outputs are often natural but less reliable on requested controls, and TraceGen combines strong control recovery with explicit reports of unsupported configurations. Together, these results suggest that TraceGen provides a compact, auditable realisation interface for controlled sentence variation from explicit frame–control records.

1 Introduction

Generated text is often used to test, augment, and analyse natural language processing (NLP) systems. Here, we study a narrower controlled-generation setting: the generator receives an explicit record of what should be expressed and a set of visible wording controls, and it should change only the requested wording. This matters because a fluent sentence no longer tests the intended condition if it adds an event, drops a participant, or realises a different request form.

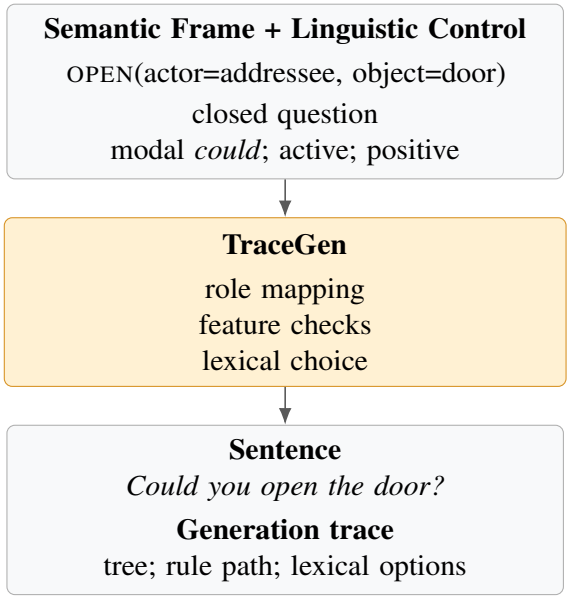

Figure 1: TraceGen keeps the event fixed while changing the requested form: a semantic frame and linguistic controls are realised as a sentence, with a trace of the grammar choices used to build it.

To capture the fixed meaning, we represent it as a semantic frame: a compact description of the situation to express, including the event and its participants. For example, *Open the door*, *Could you open the door?*, and *Would you mind opening the door?* preserve the same OPEN frame, with the addressee acting on the door. They differ in form: a direct command, a modal question, and a more indirect request. By contrast, *Could you open the door and turn on the light?* adds a second event. It may be natural English, but it is not a controlled variant of the original frame.

This contrast exposes a practical trade-off. Templates are inspectable because much of the output is written in advance, but interacting controls require more patterns. A prompted LLM, by contrast, is flexible and often fluent, but it does not expose a derivation and may silently choose another linguistic configuration, omit a slot, or add content. The relevant NLG requirement is therefore not only flu-

ency, but whether the output expresses the input it was asked to express (Reiter and Dale, 2000; Dušek and Kasner, 2020).

To address this controlled-realisation problem, we introduce TraceGen (Figure 1), a grammar-guided English realiser for this setting. TraceGen uses semantic frames to fix the intended situation and linguistic configurations to choose the wording. A lightweight grammar checks compatibility between frame and controls, builds the sentence, and records the choices made during generation. The implementation uses the category/function distinction from *The Cambridge Grammar of the English Language* (CGEL) (Huddleston and Pullum, 2002): a semantic role can remain fixed while its surface function changes, as in active/passive alternation. CGELBank demonstrates that CGEL-style distinctions can be made operational in NLP annotation (Reynolds et al., 2023). TraceGen applies the same distinction to generation: semantic roles remain fixed in the frame, while grammatical functions vary with the requested controls. The result is a controlled realiser that produces both a sentence and a trace of the choices that licensed it.

We therefore ask whether a grammar-guided realiser can generate controlled English variants from semantic frames while preserving the intended frame, realising the requested linguistic controls, and exposing generation choices. We make three contributions. First, we define a frame/control contract for controlled sentence realisation, separating what must remain semantically fixed from the wording choices that may vary. Second, we introduce TraceGen, a grammar-guided realiser that builds sentences from this contract and returns derivation traces. Third, we evaluate TraceGen against reusable templates and LLM baselines under the same contract. The evaluation treats each output as a contract item rather than as a free-standing sentence: it must preserve the intended frame, realise the applicable controls, and retain enough provenance to make control outcomes inspectable.

2 Related Work

This setting connects three strands of NLG work: synthetic text generation, structured-input generation, and template-based control. LLM workflows now support generation, annotation, curation, and evaluation, while earlier augmentation methods used lexical replacement, contextual augmentation, and grammar-based generation (Feng et al., 2021; Chen et al., 2023; Hahn et al., 2023; Long et al., 2024; Tan et al., 2024). In parallel, structured-input benchmarks such as WebNLG, ToTTo, and the E2E NLG Challenge test faithfulness to records and semantic inputs, but they do not usually require explicit recovery of clause form, request strategy, voice, valency, or information packaging (Gardent et al., 2017; Parikh et al., 2020; Dušek et al., 2020). Templates provide inspectable control, but interacting controls often require many construction patterns (van Deemter et al., 2005; Puzikov and Gurevych, 2018).

Within this landscape, TraceGen follows the classical NLG separation between deciding what to say and realising how to say it (Reiter and Dale, 2000). Its nearest alternatives differ in what they take as input and what they treat as success: templates fill string patterns, practical realisers such as SimpleNLG realise linguistic specifications, precision grammars generate from denser semantic representations, AMR-to-text systems verbalise semantic graphs, and LLM prompting offers soft control through instructions. Earlier realisers and surface-realisation shared tasks show that explicit realisation can control syntactic form from linguistic inputs (Elhadad, 1991; Elhadad and Robin, 1996; Bateman, 1997; Lavoie and Rambow, 1997; Langkilde and Knight, 1998; White, 2006; Gatt and Reiter, 2009; Belz et al., 2011; Mille et al., 2018). Unlike broad surface-realisation tasks, our input is a compact task-level frame plus explicit wording controls, and the evaluation scores exact recovery of those controls rather than general grammaticality or similarity to references. Precision grammars and MRS-based generation provide broader semantic coverage, but typically require denser input specifications than the lightweight frame/control records used here (Copestake and Flickinger, 2000; Copestake et al., 2005; Carroll and Oepen, 2005). AMR-to-text is a neighbouring task: AMR systems verbalise semantic graphs, whereas we fix a task-level semantic frame and test recovery of an explicit linguistic configuration (Banarescu et al., 2013; Bevilacqua et al., 2021; Ribeiro et al., 2021).

At the neural end of the design space, controllable generation and constrained decoding are complementary routes to control. The LLM baselines in this paper test direct recovery from the shared frame/control record; lexically constrained decoding, predicate-logic constraints, prompt controls, LLM-based control methods, and automated correction pipelines can steer or repair neural outputs

in settings with search, validation, or post-hoc correction (Hokamp and Liu, 2017; Post and Vilar, 2018; Lu et al., 2021; Prabhumoye et al., 2020; Zhang et al., 2023; Liang et al., 2024; Pan et al., 2024). Our evaluation follows semantic-accuracy and error-reporting work in NLG: generated text should express the intended input, avoid unsupported additions, and report concrete failure modes (Wiseman et al., 2017; Dušek and Kasner, 2020; Thomson and Reiter, 2020; Thomson et al., 2023; van Miltenburg et al., 2021). Because generic NLG metrics may not test requested controls, deterministic frame/control checks are the primary automatic evidence; repetition, runtime, and trace/source diagnostics are supporting evidence (Novikova et al., 2017).

Finally, the linguistic representation draws on frame semantics, constructional form–meaning pairings, and CGEL-style syntax. Frame semantics treats events and participants as structured inputs; construction grammar links form choices with meaning-preserving variants; CGEL supplies the category/function terms used by TraceGen; and CGELBank shows that such labels can be operationalised in NLP annotation (Fillmore, 1982; Goldberg, 1995; Huddleston and Pullum, 2002; Reynolds et al., 2023). Speech-act and politeness labels are used to denote visible surface configurations rather than speaker intention or social force (Austin, 1962; Searle, 1969; Brown and Levinson, 1987). Feature checking uses a simple unification-style compatibility test, rejecting rule choices that conflict with the requested control values (e.g., `modality=could`) (Shieber, 1986; Abney, 1997).

3 The Frame–Control Contract

Controlled realisation separates what is to be expressed from how it is to be realised. In our setting, each input row contains two constraints: a semantic frame and a set of requested wording controls.

3.1 Semantic Frame Constraints

A semantic frame specifies the content that must remain stable across controlled variants. It records the event to express and the entities or modifiers that participate in that event. In the running example, the frame does not require *the door* to remain the grammatical Object. Rather, it requires an OPEN event in which the addressee is the actor, the door is the patient, and only one event is expressed. The surface grammar may change under controls such as request strategy or voice: *Could you open the door?* realises the door as Object, whereas a passive variant may realise it as Subject. Both preserve the frame because the door continues to fill the same semantic role. By contrast, *Could you open the door and turn on the light?* adds a second event, and *Could you open the window?* replaces the patient. Both are therefore outside the contract. Formally, we represent a frame as

$$F = \langle p, n, R, S, \Lambda, \nu, B \rangle,$$

where p is the predicate, n is the licensed event count, R maps semantic roles to required fillers that the frame always needs, S records slot fillers that must be preserved when present, Λ is the licensed lexical material, ν is the default valency, and B is the set of forbidden additions. For the running example, $p = \text{OPEN}$, $n = 1$, $S = \varnothing$, and $\nu =$ *transitive*. The role map R_{OPEN} maps ACTOR to *addressee* and PATIENT to *door*.

The frame therefore makes explicit that *door* is a required semantic-role filler, not a fixed grammatical function. Voice and request-form controls may change where that filler appears in the sentence, but they may not change the event, replace the patient, or add an unlicensed event.

This paper uses lightweight, task-level frames rather than full FrameNet or PropBank annotation (Baker et al., 1998; Palmer et al., 2005). Each frame records a predicate, required participants, optional slots, lexical material, default valency, and forbidden additions.

3.2 Requested Wording Controls

Wording controls specify the visible linguistic choices that the generator must realise for a given row. Each row stores the requested control values and an applicability mask. The mask identifies which controls are meaningful for that row and therefore enter the score.

In this work, the control dimensions include **clause type**, distinguishing statements, commands, questions, and exclamations; **request strategy**, distinguishing direct commands such as *Open the door* from modal questions and indirect *mind* requests; and **modality**, recording auxiliaries such as *can*, *could*, *would*, and *should*. They also include **voice**, so that a participant can change surface function without changing its semantic role; **polarity**, distinguishing positive and negative realisations; **valency**, covering transitive, double-object,

PP-complement, and particle-verb patterns; and **information packaging**, covering choices such as vocatives, fronted time or location, and supplements. The appendix gives the complete machine-readable option set, applicability masks, and file manifest for the evaluation plans (Appendix E).

3.3 Success Criteria

Together, the two constraints define success. Each evaluation row contains a semantic frame, a requested-control record, and an applicability mask, written as $i = \langle F, C, A \rangle$. The system returns one sentence y, plus source or trace metadata where available. The output succeeds only when $\mathrm{preserve}(y, F)$ and $\mathrm{controls}(y, C, A)$ both hold: the sentence must preserve the frame and realise every control selected by the applicability mask.

Thus, a sentence can be fluent and still fail. If the target is a *could* question, *Please open the door.* preserves the OPEN event but fails the requested clause type, modality, and request strategy. Conversely, a grammar-guided system may reject a configuration its grammar cannot realise. This makes non-coverage inspectable: the system reports the configuration as unsupported rather than silently returning a different sentence.

Applicability matters because not every control is meaningful for every plan. A request strategy is scored for directive rows, but not for ordinary sentiment assertions. We therefore evaluate only the controls selected by the applicability mask.

4 TraceGen

CGELBank (Reynolds et al., 2023) is a human-annotated treebank of English based on the syntactic framework of The Cambridge Grammar of the English Language (CGEL). It provides a formalized annotation scheme for representing CGEL-style constituent structure, including both syntactic categories and grammatical functions. Following CGELBank and the CGELBank annotation manual (Reynolds et al., 2025), we use CGELBank-style syntactic analysis as the target representation for generation. TraceGen adds the generative layer: it builds sentences from semantic frames and requested wording controls. The process has three stages. First, the semantic frame is converted into one or more blueprints. Second, a stochastic unification-based grammar (Johnson et al., 1999) expands a compatible blueprint into a syntactic structure. Third, lexical entries fill the word positions in that structure. A companion grammar manual, included in the supplementary material, gives the full rule inventory, feature constraints, compatibility tables, and exception policy; Appendix A gives the trace schema and paper-facing rule notation.

4.1 From Frames to Blueprints

The first stage converts a semantic frame into one or more blueprints. A frame fixes the event and its role fillers, but not their surface grammatical functions. Thus, in the running OPEN example, the addressee remains the actor and the door remains the patient whether the sentence is realised as an active clause or a passive clause.

A blueprint gives one possible grammatical interpretation of that frame. We write it as $B = \langle M, K \rangle$, where M maps frame roles to grammatical functions and K records requirements that the later syntactic structure must satisfy. For an active OPEN blueprint, $M(\text{ACTOR}) = \text{SUBJECT}$ and $M(\text{PATIENT}) = \text{OBJECT}$; for a passive blueprint, $M(\text{PATIENT}) = \text{SUBJECT}$, with the actor omitted or realised in a *by*-phrase. Therefore, controls such as voice select among compatible blueprints rather than altering the semantic frame.

This also separates blueprints from sentence templates. Whereas a template fixes part of the final string, as in *Could you {verb} {object}?*, a blueprint fixes only how semantic roles may enter a grammatical structure. As a result, later stages remain responsible for the remaining realisation choices, such as clause type, lexical choice, and polarity.

4.2 From Blueprints to Syntactic Structure

At this point, a blueprint specifies how frame roles map to grammatical functions, but not the complete sentence structure. The second stage therefore builds a syntactic structure for that mapping. TraceGen does this with a stochastic unification-based grammar: grammar rules propose possible structures, and features check whether each structure fits the requested controls and lexical choices.

Figure 2 shows the syntactic structure of a simple active clause. The sentence is divided into a noun phrase (NP) functioning as Subject and a verb phrase (VP) functioning as Predicate. Inside the VP, the verb functions as Predicator and a second NP functions as Object. The figure therefore illustrates the category/function distinction: a category says what kind of expression something is, such as NP or VP, while a function says what job it per-

forms, such as Subject or Object. Thus, *the robot* and *the door* are both NPs, but they do different jobs in the clause.

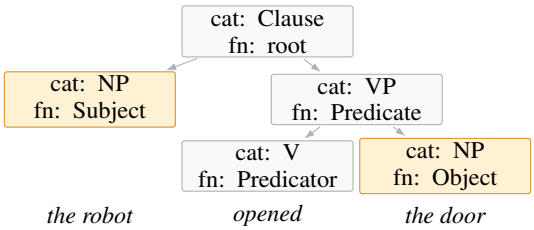

Figure 2: TraceGen derivation sketch illustrating syntactic structure and the category/function distinction. The same category, NP, can realise different functions, here Subject and Object.

This distinction links the syntactic structure back to the blueprint. The blueprint can map the same semantic role to different grammatical functions under different controls. For example, in an active clause, the patient of OPEN appears as Object, as in *the robot opened the door*. Under a passive control, the same patient appears as Subject, as in *the door was opened*. The semantic role stays fixed, but its grammatical function changes.

However, structure alone is not enough: the grammar must also check whether the pieces can combine. This is the role of features. A feature is a named grammatical property, such as number, verb form, or voice, that acts as a compatibility label. For example, NP $\rightarrow$ Det Nom says that a noun phrase can contain a determiner followed by a nominal. A feature-bearing version, NP[NUM=X] $\rightarrow$ Det[NUM=X] Nom[NUM=X], adds a number check: the determiner and nominal must share the same number value. This accepts *these cats*, where both parts are plural, and rejects the ungrammatical phrase *this cats*, where *this* is singular but *cats* is plural.

The same kind of check guides generation in TraceGen. A modal auxiliary such as *could* requires a plain-form verb, so *could open* is accepted but *could opening* is rejected. Similarly, a passive realisation requires both a passive voice value and a blueprint in which the patient can be realised as Subject. Therefore, TraceGen keeps a candidate structure only when the blueprint, grammar rules, lexical entries, and requested controls fit together.

4.3 From Syntactic Structure to Sentence

The final stage fills the syntactic structure with lexical entries and returns a sentence. Building on Figure 2, the structure provides positions such as Subject NP, Predicator, and Object NP; the lexicon then supplies compatible entries such as *the robot*, *opened*, and *the door*. These entries are typed rather than plain strings: they record surface form together with grammatical information, such as category, verb form, valency, and compatible frame roles. As a result, a frame-level concept such as OPEN or DOOR may have several recorded lexical realisations, including synonyms where available, but the selected entries must still fit the syntactic requirements. For example, *could* requires a plain-form verb, so *could open* is accepted but *could opened* is not. Finally, to make each generated sentence auditable, TraceGen stores the main generation decisions in a *trace*: the selected blueprint, grammar rules, feature checks, and lexical entries.

5 Experimental Setup

To assess whether TraceGen and other generator families can satisfy the frame–control contract, we evaluate all systems against the three requirements: (i) preserving the semantic frame, (ii) realising the applicable wording controls, and (iii) retaining enough provenance for audit.

5.1 Evaluation Contracts

This comparison requires a fixed set of frame–control contracts, so all systems use the same 160 contracts. Each contract pairs one semantic frame with one linguistic configuration and an applicability mask specifying which controls are scored. The contracts cover four scenario families: 48 HuRIC robot-command contracts, 40 SNIPS intent-query contracts, 40 SST-2 sentiment/evaluation contracts, and 32 Symptom2Disease symptom-report contracts (Socher et al., 2013; Wang et al., 2019; Bastianelli et al., 2014; Barman, 2021; Coucke et al., 2018). For example, a HuRIC contract may request a negative indirect command such as *Would you mind not turning on the lamp?*, while an SST-2 contract may request an exclamative evaluation such as *What a disappointing ending!*. The contracts are fixed before generation and are diagnostic rather than a broad generalisation benchmark: they stress the grammar patterns implemented in TraceGen while keeping unsupported configurations visible, so impossible or unsupported requests are counted as explicit failures.

5.2 Systems Compared

Using the fixed evaluation contracts, we compare seven realisers. The template realisers are Template-5 and Template-10, which use 5 and 10

reusable templates per seed, respectively. They test what a small pattern bank can recover, for example by filling a pattern such as *Could you {verb} {object}?* to produce *Could you open the door?*. The grammar-guided realiser is TraceGen, which tests explicit realisation from the frame–control contract. The LLM realisers use a one-call, prompt-only protocol: GPT-OSS-20B with low and medium reasoning effort, and Qwen3.6 in non-thinking and thinking modes (OpenAI, 2025; Qwen Team, 2025, 2026). Each LLM realiser receives the frame, requested controls, and applicability mask, and returns one sentence. Stronger LLM pipelines, such as few-shot prompting, validator/retry loops, constrained decoding, and repair-based generation, are outside the scope of this comparison, which tests direct recovery from the shared contract. Appendix D gives the template details, and Appendix C gives the LLM prompt and decoding settings.

5.3 Metrics and Verification Audit

We evaluate the frame–control contract with two complementary methods: automatic checks and an author-led verification audit. The automatic checks score two independent outcomes. *Frame preservation* measures semantic alignment: whether the sentence keeps the intended event, required participants, required slots, and event count. *Exact configuration* measures linguistic control: whether every applicable requested control is realised, including clause type, request strategy, modality, voice, polarity, valency, and information packaging. Thus, a sentence can preserve the frame while failing exact configuration, or realise the requested form while failing to preserve the frame. The checker is a rule-based scorer that compares each cleaned sentence with the requested controls and applicability mask using deterministic surface tests. These tests recover controls from observable cues, such as modal auxiliaries, negation, passive marking, word order, punctuation, request-form patterns, and preserved frame/slot fillers, rather than from a TraceGen trace, template id, or LLM metadata. Appendix E.7 reports implementation details. In addition, two author annotators judge whether each generated sentence satisfies the same contract without seeing which realiser produced it (van der Lee et al., 2019, 2021). The primary human metric is *manual faithful*: the sentence must preserve the frame, keep required participants and slots, realise the requested form and request strategy where applicable, avoid unsupported additions and metadata leakage, and be usable as one sentence. The audit also reports naturalness and rigid wording as diagnostics, which help interpret failures but are not the primary success criterion. Details of the automatic checks are reported in Appendix G; details of the audit design, codebook, agreement table, and export schema are reported in Appendix H.

Metric	T5	T10	TG	GL	GM	QN	QT
Exact cfg.	64.4	56.9	**91.9**	37.5	39.2	38.1	50.5
Frame	**100.0**	**100.0**	91.9	96.1	96.8	89.2	89.2

Table 1: Automatic diagnostics (%; higher is better). T5/T10 are Template-5/10; TG is TraceGen; GL/GM are GPT-OSS low/medium; QN/QT are Qwen non-thinking/thinking. Each seed contributes 160 counted contracts per realiser. Values are five-seed means.

6 Results and Discussion

The results support three main conclusions. First, TraceGen is the most reliable realiser for exact-control recovery under the automatic frame–control contract, reaching 91.9% exact-configuration recovery under strict scoring that counts explicit coverage failures. Second, templates preserve frames but are rigid, while LLMs are more natural but drift from requested controls. Finally, the remaining TraceGen errors are explicit coverage gaps, making missing grammar capabilities visible rather than hidden in fluent substitutions.

6.1 TraceGen Recovers Requested Controls Most Reliably

The main automatic result is about control recovery. Each realiser is evaluated on the same 160 contracts across five seeds, giving 5,600 counted records in total. Explicit failures are counted as failures, so the scores measure success over the pre-specified contracts rather than over successful generations only. Table 1 reports five-seed mean percentages. A $\downarrow$ marks metrics where lower values are better; otherwise, higher values are better. **Bold** marks the best displayed value, and underlining marks the second-best distinct displayed value.

The key row is *Exact configuration*, because it asks whether every applicable requested control was realised. On this measure, TraceGen reaches 91.9%, clearly above Template-5 (64.4%), Template-10 (56.9%), and the LLM baselines (37.5–50.5%). This gap shows that direct fluency is not enough for controlled realisation: the LLMs often produce acceptable English, and some preserve

	Faith.	Frame	Form	Req.	Nat.	Rigid ↓
T5	**75**	**100**	**96**	**100**	54	21
T10	**75**	**100**	**96**	**100**	46	17
TG	71	88	92	**100**	71	4
GL	29	92	38	58	79	**0**
GM	29	88	38	75	74	13
QN	21	92	21	75	79	4
QT	46	92	71	92	**92**	4

Table 2: Final consensus human verification audit for the seven primary realisers, with entries reported as percentages ($N = 24$ outputs per realiser). T5/T10 are Template-5/10; TG is TraceGen; GL/GM are GPT-OSS low/medium; QN/QT are Qwen non-thinking/thinking. Higher is better for all metrics except *Rigid*, where lower is better.

the frame well, but they less reliably recover the requested wording controls.

The frame-preservation row should therefore be read separately. Template-5 and Template-10 preserve all frames under the automatic checker, and GPT-OSS medium reaches 96.8%, but these systems still recover fewer requested controls. Conversely, TraceGen's 91.9% frame-preservation score reflects 735 successful traced generations and 65 explicit coverage failures over 800 attempts. Thus, the lower score reflects explicit coverage failures rather than silent control substitutions observed in LLM baselines.

6.2 Human Audit Separates Faithfulness, Control, and Naturalness

The automatic exact-configuration score captures control recovery, but it does not by itself explain whether sentences sound natural or rigid. For that reason, we also run a verification audit over 168 consensus outputs: 24 matched contracts crossed with the seven realisers. Two annotators first judged the outputs independently and then resolved disagreements by consensus. Because each realiser has only $N = 24$ audited outputs, the percentages in Table 2 are descriptive contract-verification evidence rather than population estimates.

The audit shows a trade-off that the automatic table compresses. The template realisers are the most strictly faithful on the audited subset, but they are also less natural and more rigid. TraceGen is close in strict faithfulness, recovers requested form and request strategy reliably, and has low rigidity. The LLM realisers are usually more natural, especially Qwen thinking, but they more often drift from the requested form or request strategy.

TraceGen's row illustrates this profile: it has 71% strict manual faithfulness, 88% frame and role/slot preservation, 92% form recovery, 100% request-strategy recovery on applicable request rows, 71% naturalness, and only 4% rigidity. Thus, TraceGen is not the most natural realiser, but it is the most control-oriented one. By contrast, Qwen thinking has the highest naturalness score (92%) and strong request-strategy recovery (92%), yet lower strict faithfulness (46%). This confirms the main interpretation: naturalness and contract satisfaction are related but not equivalent.

Unsupported additions are not the main source of failure in the audit. No-unsupported rates are high across realisers: 96% for both template realisers, 92% for TraceGen and GPT-OSS low, 83% for GPT-OSS medium, and 88% for both Qwen realisers. The main distinction is instead whether the output realises the requested linguistic configuration while preserving the frame.

6.3 Exact Configuration Is the Headline Control Metric

The difference between frame preservation and exact configuration is easiest to see at sentence level. A sentence can keep the event and participants but still fail the requested controls. For example, if the contract asks for *Would you mind not turning on the floor lamp in the bedroom?*, then *Would you not turn on the floor lamp in the bedroom?* keeps the lamp, the bedroom, the modal question, and the negation. It still fails because it changes the requested *mind* request strategy into another request form. This is why exact configuration, rather than frame preservation alone, is the headline control metric. Furthermore, the same logic applies to full audit rows. In `huric_041`, the frame is TURN_ON(agent=robot, patient=floor lamp) with location=bedroom, and the applicable controls require a closed interrogative directive with *would*, negative polarity, active voice, particle-verb valency, the *mind* request strategy, and neutral packaging. The output *Would you mind not turning on the floor lamp in the bedroom?* succeeds because it keeps the required frame material and realises the requested negative *mind* interrogative. In contrast, *Would you not turn on the floor lamp in the bedroom?* keeps much of the same content but fails the requested strategy.

A non-request example shows why the applicability mask is also necessary. In `ssttwo_030`, the frame is REJECT(reviewer=reviewer, pa-

tient=ending) with aspect=resolution, and the applicable controls require an exclamative expressive sentence, active voice, affirmative polarity, transitive valency, and supplement packaging. Request strategy and modality are not applicable, so they do not affect the score. Therefore, *The reviewer rejects the ending* is not wrong because it lacks a request strategy; it is wrong because it drops the resolution aspect and the requested exclamative form. These are control and frame failures, not fluency failures.

6.4 Coverage Failures Identify Grammar Gaps

For TraceGen, the remaining failures are explicit grammar-coverage failures rather than fluent substitutions. Across five seeds, 735/800 TraceGen records contain grammar traces and 65 are explicit coverage failures. These failures are concentrated in HuRIC: 60 require double-object profiles with both recipient and object support, and 5 require fronted-location profiles when the frame lacks a location or destination slot. In scenario terms, TraceGen succeeds on 175/240 HuRIC attempts, or 72.9%, while SNIPS, SST-2, and Symptom2Disease have no TraceGen coverage failures in the five-seed evaluation.

This pattern is useful because it identifies the grammar work still required. A visible refusal points to a missing grammar capability, such as double-object support. A fluent substitution from another realiser may look better at first glance, but it can hide the fact that the requested configuration was not actually realised.

6.5 Failure Modes Differ by Realiser Family

The realiser families fail in different ways. Templates preserve frames well because their reusable patterns were authored for compatible cases; however, the same finite pattern bank limits variation and misses some requested forms. LLM baselines often produce fluent and natural sentences; however, they lose exact configuration when request strategy, valency, or information packaging must be preserved. TraceGen either satisfies the implemented grammar contract or returns an explicit coverage record.

The row-level labels make these differences inspectable. In `huric_041`, *Would you not turn on the floor lamp in the bedroom?* is a wrong-request-strategy error: it keeps the frame, modal, and negation, but not the requested *mind* strategy. In `ssttwo_030`, *At the end, the reviewer rejects the ending* is an aspect-drop error: the reviewer and ending remain, but the resolution aspect is missing. In `huric_030`, *The water bottle should be brought to the office desk for Marta* is a wrong-valency error: the transfer event remains, but the requested role configuration is not realised. Finally, `huric_011` is an explicit coverage refusal, *[GENERATION FAILURE] unsupported double-object valency*. This is not a usable sentence, but it exposes the unsupported configuration instead of returning an easier substitute.

6.6 Traces Explain Successes and Refusals

Finally, traces are used for interpretation, not for scoring. The automatic checker infers controls from the output sentence, and human annotators see the frame–control contract and sentence rather than the TraceGen trace. After scoring, however, the trace explains why a TraceGen sentence was accepted by the grammar or why a requested configuration was refused. Thus, traces turn a control failure into an inspectable grammar or lexicon gap rather than a hidden substitution. Appendix E and Appendix J document the artefact structure used to inspect the frame–control contracts and trace records.

7 Conclusion

This paper introduced TraceGen, a grammar-guided realiser for controlled English generation from semantic frames and explicit wording controls. The central design separates the content that must remain fixed from the linguistic choices that may vary. A semantic frame specifies the event, participants, slots, and forbidden additions; the control record specifies the requested surface configuration; and the trace records the grammar choices used to produce a successful output.

The results show that controlled realisation is not the same as fluent sentence generation. Templates preserve many covered frames but offer limited variation. One-call prompt-only LLM baselines often produce natural sentences, but less reliably recover the requested controls. TraceGen gives the strongest exact-configuration recovery under the automatic contract, while keeping unsupported configurations visible as coverage failures. This makes it useful for settings where the intended meaning and linguistic controls are known in advance, and where failures should be auditable rather than hidden in fluent substitutions.

Limitations

This study is deliberately narrow. The evaluation uses a compact English frame/control grid, so the results should not be read as evidence of broad English coverage or open-domain generation ability. TraceGen provides structural guarantees only within the implemented grammar, lexicon, and compatibility rules. Successful outputs are licensed by those resources and return traces, but semantic adequacy, naturalness, and social appropriateness still require external checking.

The automatic checker is a reproducible diagnostic rather than an oracle. It uses deterministic surface tests and may reject acceptable paraphrases that depart from the lexical record. The human audit provides useful verification evidence, but it is author-led, uses two annotators, and reports per-lane rates over only $N = 24$ outputs. These rates should therefore be interpreted descriptively rather than as population estimates.

The comparison also uses a limited LLM protocol: one call, one output, no few-shot examples, no validator-guided retry loop, no constrained decoding, and no post-hoc repair. Stronger LLM pipelines may reduce some control errors. Similarly, broader grammar coverage, richer lexical inventories, and logged template-authoring effort could change the practical trade-offs reported here. The current results should therefore be understood as evidence for this frame/control contract and implementation, not as a final ranking of all possible realisation systems.

Ethical Considerations and Data Use

The released frame/control plans and generated outputs are synthetic records derived from cited scenario families. They are intended for reproducing the controlled-generation evaluation, inspecting frame/control failures, and testing annotation procedures. They should not be treated as real user language, and downstream use should preserve the synthetic provenance metadata attached to each row (Bender and Friedman, 2018; Rogers et al., 2021).

The Symptom2Disease-derived examples are used only as synthetic symptom-report frames for generation experiments. They are not clinical cases, diagnoses, medical advice, or evidence about real patients. The artefact should not be used for clinical decision support or for claims about real health behaviour.

The verification audit used generated outputs and synthetic frame/control records only. It was carried out by the paper authors; no external annotators, crowdworkers, students, or personal data were involved. The release follows the stated source-resource terms: third-party datasets are cited but not relicensed, and only permitted derived records, generated outputs, code, prompts, scoring scripts, validation files, and audit artefacts are included. Appendix J.1 gives the data statement and intended-use constraints; Appendix H reports the audit protocol; Appendix F reports compute; and Appendix K discloses AI-assistance use.

References

Steven P. Abney. 1997. Stochastic attribute-value grammars. *Computational Linguistics*, 23(4):597–618.

Ron Artstein and Massimo Poesio. 2008. Survey article: Inter-coder agreement for computational linguistics. *Computational Linguistics*, 34(4):555–596.

J. L. Austin. 1962. *How to Do Things with Words*. Clarendon Press, Oxford. Edited by J. O. Urmson.

Collin F. Baker, Charles J. Fillmore, and John B. Lowe. 1998. The Berkeley FrameNet project. In *36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics, Volume 1*, pages 86–90, Montreal, Quebec, Canada. Association for Computational Linguistics.

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. Abstract Meaning Representation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, pages 178–186, Sofia, Bulgaria. Association for Computational Linguistics.

Niyarr Barman. 2021. Symptom2Disease: Diseases and Natural Language Symptom Descriptions. `https://www.kaggle.com/datasets/niyarrbarman/symptom2disease`. Kaggle dataset. Accessed: 2026-05-22.

Emanuele Bastianelli, Giuseppe Castellucci, Danilo Croce, Luca Iocchi, Roberto Basili, and Daniele Nardi. 2014. HuRIC: A human robot interaction corpus. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation*, pages 4519–4526.

John A. Bateman. 1997. Enabling technology for multilingual natural language generation: The KPML development environment. *Natural Language Engineering*, 3(1):15–55.

Anja Belz, Michael White, Dominic Espinosa, Eric Kow, Deirdre Hogan, and Amanda Stent. 2011. The first surface realisation shared task: Overview and evaluation results. In *Proceedings of the 13th European Workshop on Natural Language Generation*, pages 217–226, Nancy, France. Association for Computational Linguistics.

Emily M. Bender and Batya Friedman. 2018. Data statements for natural language processing: Toward mitigating system bias and enabling better science. *Transactions of the Association for Computational Linguistics*, 6:587–604.

Michele Bevilacqua, Rexhina Blloshmi, and Roberto Navigli. 2021. One SPRING to rule them both: Symmetric AMR semantic parsing and generation without a complex pipeline. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(14):12564–12573.

Penelope Brown and Stephen C. Levinson. 1987. *Politeness: Some Universals in Language Usage*. Cambridge University Press.

Ted Byrt, Janet Bishop, and John B. Carlin. 1993. Bias, prevalence and kappa. *Journal of Clinical Epidemiology*, 46(5):423–429.

John Carroll and Stephan Oepen. 2005. High efficiency realization for a wide-coverage unification grammar. In *Proceedings of the Second International Joint Conference on Natural Language Processing*, pages 165–176, Jeju Island, Korea.

Jiaao Chen, Derek Tam, Colin Raffel, Mohit Bansal, and Diyi Yang. 2023. An empirical survey of data augmentation for limited data learning in NLP. *Transactions of the Association for Computational Linguistics*, 11:191–211.

Domenic V. Cicchetti and Alvan R. Feinstein. 1990. High agreement but low kappa: Ii. resolving the paradoxes. *Journal of Clinical Epidemiology*, 43(6):551–558.

Jacob Cohen. 1960. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(1):37–46.

Ann Copestake and Dan Flickinger. 2000. An open source grammar development environment and broad-coverage English grammar using HPSG. In *Proceedings of the Second International Conference on Language Resources and Evaluation (LREC'00)*, Athens, Greece. European Language Resources Association.

Ann Copestake, Dan Flickinger, Carl Pollard, and Ivan A. Sag. 2005. Minimal recursion semantics: An introduction. *Research on Language and Computation*, 3(2–3):281–332.

Alice Coucke, Alaa Saade, Adrien Ball, Théodore Bluche, Alexandre Caulier, David Leroy, Clément Doumouro, Thibault Gisselbrecht, Francesco Caltagirone, Thibaut Lavril, Maël Primet, and Joseph Dureau. 2018. Snips voice platform: An embedded spoken language understanding system for private-by-design voice interfaces. *Preprint*, arXiv:1805.10190.

Ondřej Dušek and Zdeněk Kasner. 2020. Evaluating semantic accuracy of data-to-text generation with natural language inference. In *Proceedings of the 13th International Conference on Natural Language Generation*, pages 131–137.

Ondřej Dušek, Jekaterina Novikova, and Verena Rieser. 2020. Evaluating the state-of-the-art of end-to-end natural language generation: The E2E NLG challenge. *Computer Speech & Language*, 59:123–156.

Michael Elhadad. 1991. FUF: The universal unifier – user manual, version 5.0. Technical Report CUCS-038-91, Columbia University.

Michael Elhadad and Jacques Robin. 1996. An overview of SURGE: A reusable comprehensive syntactic realization component. In *Proceedings of the 8th International Natural Language Generation Workshop*, pages 1–4.

Steven Y. Feng, Varun Gangal, Jason Wei, Sarath Chandar, Soroush Vosoughi, Teruko Mitamura, and Eduard Hovy. 2021. A survey of data augmentation approaches for NLP. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 968–988.

Charles J. Fillmore. 1982. Frame semantics. In The Linguistic Society of Korea, editor, *Linguistics in the Morning Calm*, pages 111–137. Hanshin Publishing, Seoul.

Claire Gardent, Anastasia Shimorina, Shashi Narayan, and Laura Perez-Beltrachini. 2017. The WebNLG challenge: Generating text from RDF data. In *Proceedings of the 10th International Conference on Natural Language Generation*, pages 124–133, Santiago de Compostela, Spain. Association for Computational Linguistics.

Albert Gatt and Ehud Reiter. 2009. SimpleNLG: A realisation engine for practical applications. In *Proceedings of the 12th European Workshop on Natural Language Generation*, pages 90–93.

Adele E. Goldberg. 1995. *Constructions: A Construction Grammar Approach to Argument Structure*. University of Chicago Press, Chicago.

Joonghyuk Hahn, Hyunjoon Cheon, Elizabeth Orwig, Su-Hyeon Kim, Sang-Ki Ko, and Yo-Sub Han. 2023. GDA: Grammar-based data augmentation for text classification using slot information. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 7291–7306.

Chris Hokamp and Qun Liu. 2017. Lexically constrained decoding for sequence generation using grid beam search. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1535–1546, Vancouver, Canada. Association for Computational Linguistics.

Rodney Huddleston and Geoffrey K. Pullum. 2002. *The Cambridge Grammar of the English Language*. Cambridge University Press.

Mark Johnson, Stuart Geman, Stephen Canon, Zhiyi Chi, and Stefan Riezler. 1999. Estimators for stochastic "unification-based" grammars. In *Proceedings of ACL 1999*, pages 535–541.

Irene Langkilde and Kevin Knight. 1998. Generation that exploits corpus-based statistical knowledge. In *36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics, Volume 1*, pages 704–710, Montreal, Quebec, Canada. Association for Computational Linguistics.

Benoit Lavoie and Owen Rambow. 1997. A fast and portable realizer for text generation systems. In *Fifth Conference on Applied Natural Language Processing*, pages 265–268, Washington, DC, USA. Association for Computational Linguistics.

Jiwei Li, Michel Galley, Chris Brockett, Jianfeng Gao, and Bill Dolan. 2016. A diversity-promoting objective function for neural conversation models. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 110–119, San Diego, California. Association for Computational Linguistics.

Xun Liang, Hanyu Wang, Yezhaohui Wang, Shichao Song, Jiawei Yang, Simin Niu, Jie Hu, Dan Liu, Shunyu Yao, Feiyu Xiong, and Zhiyu Li. 2024. Controllable text generation for large language models: A survey. *Preprint*, arXiv:2408.12599.

Lin Long, Rui Wang, Ruixuan Xiao, Junbo Zhao, Xiao Ding, Gang Chen, and Haobo Wang. 2024. On LLMs-driven synthetic data generation, curation, and evaluation: A survey. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 11065–11082, Bangkok, Thailand. Association for Computational Linguistics.

Ximing Lu, Peter West, Rowan Zellers, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. NeuroLogic decoding: (un)supervised neural text generation with predicate logic constraints. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4288–4299, Online. Association for Computational Linguistics.

Simon Mille, Anja Belz, Bernd Bohnet, Yvette Graham, Emily Pitler, and Leo Wanner. 2018. The first multilingual surface realisation shared task (SR'18): Overview and evaluation results. In *Proceedings of the First Workshop on Multilingual Surface Realisation*, pages 1–12, Melbourne, Australia. Association for Computational Linguistics.

Jekaterina Novikova, Ondřej Dušek, Amanda Cercas Curry, and Verena Rieser. 2017. Why we need new evaluation metrics for NLG. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2241–2252, Copenhagen, Denmark. Association for Computational Linguistics.

OpenAI. 2025. gpt-oss-120b and gpt-oss-20b model card. *Preprint*, arXiv:2508.10925.

Woodley Packard. 2013. ACE: The answer constraint engine. Software package.

Martha Palmer, Daniel Gildea, and Paul Kingsbury. 2005. The Proposition Bank: An annotated corpus of semantic roles. *Computational Linguistics*, 31(1):71–106.

Liangming Pan, Michael Saxon, Wenda Xu, Deepak Nathani, Xinyi Wang, and William Yang Wang. 2024. Automatically correcting large language models: Surveying the landscape of diverse automated correction strategies. *Transactions of the Association for Computational Linguistics*, 12:484–506.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Ankur Parikh, Xuezhi Wang, Sebastian Gehrmann, Manaal Faruqui, Bhuwan Dhingra, Diyi Yang, and Dipanjan Das. 2020. ToTTo: A controlled table-to-text generation dataset. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1173–1186, Online. Association for Computational Linguistics.

Matt Post and David Vilar. 2018. Fast lexically constrained decoding with dynamic beam allocation for neural machine translation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1314–1324, New Orleans, Louisiana. Association for Computational Linguistics.

Shrimai Prabhumoye, Alan W. Black, and Ruslan Salakhutdinov. 2020. Exploring controllable text generation techniques. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 1–14, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Yevgeniy Puzikov and Iryna Gurevych. 2018. E2E NLG challenge: Neural models vs. templates. In *Proceedings of the 11th International Conference on Natural Language Generation*, pages 463–471, Tilburg University, The Netherlands. Association for Computational Linguistics.

Qwen Team. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Qwen Team. 2026. Qwen3.6-35B-A3B-FP8. Hugging Face model card. FP8-quantized model weights and configuration files.

Ehud Reiter and Robert Dale. 2000. *Building Natural Language Generation Systems*. Cambridge University Press.

Brett Reynolds, Aryaman Arora, and Nathan Schneider. 2023. Unified syntactic annotation of English in the CGEL framework. In *Proceedings of the 17th Linguistic Annotation Workshop*, pages 220–234.

Brett Reynolds, Nathan Schneider, and Aryaman Arora. 2025. Cgelbank annotation manual v1.2. *Preprint*, arXiv:2305.17347.

Leonardo F. R. Ribeiro, Jonas Pfeiffer, Yue Zhang, and Iryna Gurevych. 2021. Smelting gold and silver for improved multilingual AMR-to-Text generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 742–750, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Anna Rogers, Timothy Baldwin, and Kobi Leins. 2021. 'just what do you think you're doing, dave?' a checklist for responsible data use in NLP. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 4821–4833, Punta Cana, Dominican Republic. Association for Computational Linguistics.

John R. Searle. 1969. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press.

Stuart M. Shieber. 1986. *An Introduction to Unification-Based Approaches to Grammar*. CSLI, Stanford, CA.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of EMNLP 2013*, pages 1631–1642.

Zhen Tan, Dawei Li, Song Wang, Alimohammad Beigi, Bohan Jiang, Amrita Bhattacharjee, Mansooreh Karami, Jundong Li, Lu Cheng, and Huan Liu. 2024. Large language models for data annotation and synthesis: A survey. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 930–957, Miami, Florida, USA. Association for Computational Linguistics.

Craig Thomson and Ehud Reiter. 2020. A gold standard methodology for evaluating accuracy in data-to-text systems. In *Proceedings of the 13th International Conference on Natural Language Generation*, pages 158–168, Dublin, Ireland. Association for Computational Linguistics.

Craig Thomson, Ehud Reiter, and Barkavi Sundararajan. 2023. Evaluating factual accuracy in complex data-to-text. *Computer Speech & Language*, 80:101482.

Kees van Deemter, Emiel Krahmer, and Mariët Theune. 2005. Real versus template-based natural language generation: A false opposition? *Computational Linguistics*, 31(1):15–24.

Chris van der Lee, Albert Gatt, Emiel van Miltenburg, and Emiel Krahmer. 2021. Human evaluation of automatically generated text: Current trends and best practice guidelines. *Computer Speech & Language*, 67:101151.

Chris van der Lee, Albert Gatt, Emiel van Miltenburg, Sander Wubben, and Emiel Krahmer. 2019. Best practices for the human evaluation of automatically generated text. In *Proceedings of the 12th International Conference on Natural Language Generation*, pages 355–368, Tokyo, Japan. Association for Computational Linguistics.

Emiel van Miltenburg, Miruna Clinciu, Ondřej Dušek, Dimitra Gkatzia, Stephanie Inglis, Leo Leppänen, Saad Mahamood, Emma Manning, Stephanie Schoch, Craig Thomson, and Luou Wen. 2021. Underreporting of errors in NLG output, and what to do about it. In *Proceedings of the 14th International Conference on Natural Language Generation*, pages 140–153, Aberdeen, Scotland, UK. Association for Computational Linguistics.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of ICLR 2019*.

Michael White. 2006. Efficient realization of coordinate structures in combinatory categorial grammar. *Research on Language and Computation*, 4:39–75.

Edwin B. Wilson. 1927. Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association*, 22(158):209–212.

Sam Wiseman, Stuart Shieber, and Alexander Rush. 2017. Challenges in data-to-document generation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2253–2263, Copenhagen, Denmark. Association for Computational Linguistics.

Hanqing Zhang, Haolin Song, Shaoyu Li, Ming Zhou, and Dawei Song. 2023. A survey of controllable text generation using transformer-based pre-trained language models. *ACM Computing Surveys*, 56(3).

Yaoming Zhu, Sidi Lu, Lei Zheng, Jiaxian Guo, Weinan Zhang, Jun Wang, and Yong Yu. 2018. Texygen: A benchmarking platform for text generation models. In *Proceedings of SIGIR 2018*, pages 1097–1100.

Appendix guide. Appendix A gives the grammar and trace details that support the main TraceGen description. Appendix B places TraceGen in relation to prior realisers, controlled generation, structured-input generation, and template methods. Appendix C documents the LLM prompting protocol, decoding settings, and structured-output ablation. Appendix D describes the reusable template baseline. Appendix E defines the frame/control grid, applicability masks, and automatic-checker safeguards. Appendix F records the execution contract, validation checks, compute environment, and optional adapter audit. Appendix G gives the full automatic metric breakdowns. Appendix H gives the human-audit protocol, codebook, agreement diagnostics, and export schema. Appendix I gives additional request-form examples. Appendix J describes the released software, data statement, licences, and artefact checklist. Appendix K discloses the use of AI assistants.

A Grammar and Trace Details

The main paper introduces TraceGen as a three-stage realiser: semantic frames are first mapped to blueprints, compatible blueprints are expanded by a stochastic unification-based grammar, and lexical entries realise the resulting syntactic structure as a sentence. This appendix gives the paper-facing details needed to audit that mechanism. The full rule inventory, compatibility tables, feature constraints, and exception policy are provided in the companion grammar manual included with the supplementary material.

A.1 Grammar inventory accounting

Table 3 reports the inventory-size statistics used in the paper. The table separates declared resources, such as semantic frames and control values, from observed trace evidence, such as exercised rule identifiers and distinct rule traces. This distinction matters because declared inventory size describes the input and grammar resources, whereas trace counts describe the subset of the grammar actually used by successful TraceGen outputs.

A.2 Blueprints, rules, and traces

Three internal objects are used throughout the implementation. A **blueprint** is the first grammatical interpretation of a frame: it maps semantic roles to grammatical functions under a possible realisation. For example, an active blueprint may map the actor to Subject and the patient to Object, while a passive blueprint may map the patient to Subject and express the actor optionally in a *by*-phrase. A **construction rule** then licenses a syntactic configuration, such as a transitive VP, a modal interrogative, an imperative, or a passive clause. A **trace** records the blueprint, rule sequence, feature checks, lexical choices, and final word order used to produce a successful sentence.

This separation keeps the representation compositional. The blueprint does not contain a fixed sentence string; it only states how semantic roles may enter a grammatical structure. Construction rules provide the phrase and clause shapes. Feature checks determine whether those shapes are compatible with the requested controls and lexical entries. The trace is therefore not an explanation added after generation. It is the record of the choices that built the sentence.

A.3 Feature compatibility

A feature is a named grammatical property used to license or block a rule application. In the paper-facing grammar, relevant features include clause type, modality, voice, polarity, valency, information packaging, and verb form. A rule can apply only when its own feature requirements are compatible with the requested controls, the selected blueprint, and the selected lexical entries.

For the running OPEN example, the modal-question rule checks that the requested clause type is interrogative, that the selected modal is *could*, that polarity is affirmative, and that the lexical verb can appear in the bare form after the modal. These checks license *Could you open the door?*. They reject *Open the door.*, which has the wrong clause type, and *Could you opening the door?*, where the verb form is incompatible with the modal. If no compatible rule remains, TraceGen reports the unsupported configuration rather than silently substituting an easier construction.

The compatibility test is deliberately simple. Two choices can combine only if they do not assign conflicting values to the same feature. If the requested clause type is a closed interrogative, an imperative construction is incompatible. If a modal auxiliary requires a bare verb form, a gerund-participial verb form is incompatible. This gives the grammar an inspectable mechanism for controlling interactions among request form, modality, polarity, voice, valency, and information packaging.

Statistic	Count	How it is counted
Semantic-frame prototypes	32	Rows in `semantic_frames.json` (HuRIC=8, SNIPS=8, SST-2=8, Symptom2Disease=8).
Wording-control dimensions	8	Top-level controls in `control_inventory.json`; the same eight dimensions are also stored as trace feature keys for TraceGen rows.
Declared control values	40	Sum of all `allowed_values` lists; the JSON summary reports the per-dimension counts.
Source lexical entries	106	Unique predicate lemmas, participant fillers, and slot fillers in `semantic_frames.json`; multiword expressions count as one entry.
Exercised rule identifiers	50	Unique grammar-choice identifiers appearing in `rule_trace` for successful TraceGen outputs across the five-seed evaluation artefact.
Distinct rule traces	52 per seed	Unique ordered `rule_trace` sequences among the 147 traced TraceGen rows in each seed.

Table 3: Inventory-size accounting for the reported TraceGen experiment. The source files are part of the released artefact. Coverage failures are excluded from trace-based rule counts because they do not have rule traces.

A.4 Trace record

For each successful TraceGen output, the trace stores the selected blueprint, the rule identifiers used in order, the feature checks applied, the contributed grammatical features, the role-to-function mapping, the selected lexical entries, and the final linearisation. In the running example, the frame states that the addressee opens the door and licenses one event only. The requested controls ask for an active, affirmative, transitive *could* question. The blueprint maps the actor to Subject and the patient to Object. The rule sequence licenses a modal-question clause, a transitive verb phrase, the addressee pronoun, and a definite object noun phrase. Feature checks verify that *could* selects a bare verb and that interrogative order places the auxiliary before the subject. The final output is *Could you open the door?*.

The trace is used as provenance evidence, not as a shortcut for scoring. Automatic surface-control checks and the human audit judge the generated sentence against the frame/control record. The trace explains why TraceGen licensed a successful output or why no compatible rule was available.

B Positioning Relative to Prior Systems

TraceGen sits between several neighbouring traditions in natural language generation. Classical realisers and surface-generation systems provide the historical basis for explicit realisation, but TraceGen uses a compact frame/control input and returns a derivation trace tailored to controlled sentence variation. Precision grammars and MRS-based generation offer broader semantic coverage, but they typically require denser input specifications than the lightweight frame/control records used here. Controlled natural languages share the aim of reliable control, although they usually prescribe a restricted writing language rather than generating controlled variants from a semantic frame. Structured data-to-text generation shares the faithfulness problem, but the evaluation here additionally tests recovery of clause type, request strategy, modality, voice, polarity, valency, and information packaging. Neural controllable generation is complementary: the one-call prompt-only LLM lanes test direct recovery from the shared frame/control record, while constrained decoding, validator-guided generation, retry loops, and post-hoc repair pipelines address different system settings.

This positioning is therefore not a claim that TraceGen replaces broad realisers, precision grammars, or neural generation systems. Its contribution is narrower: it defines a frame/control interface, implements a grammar-guided realiser for that interface, and evaluates whether systems satisfy the same row-level contract while preserving enough provenance for audit.

C Prompting and Decoding Settings

The released artefact contains the full prompts and prompt hashes. The paper reports the prompt contract rather than printing the full prompt text, because the contract is the controlled element shared across LLM lanes. Each one-call prompt-only LLM lane receives the same semantic frame, requested controls, applicability mask, negative constraints, and instruction to return one English sentence with no explanation. Model, reasoning effort, thinking mode, and output contract are the varying factors. The LLM identifiers are `openai/gpt-oss-20b` and `Qwen/Qwen3.6-35B-A3B-FP8` (OpenAI, 2025; Qwen Team, 2025, 2026).

For GPT-OSS-20B, the low, medium, and LLM-Structured lanes use temperature 0.7, top-p 0.95, presence penalty 0, no top-k, and a 3,072-token output budget. The two sentence-only GPT-OSS lanes differ only in the recorded `reasoning_effort`. LLM-Structured keeps medium reasoning effort and changes only the required output format. For Qwen3.6, the non-thinking lane follows the guide sampling settings: temperature 0.7, top-p 0.8, top-k 20, `min_p=0`, presence penalty 0, thinking disabled through the serving/chat-template configuration, and a 512-token run cap. The thinking lane uses temperature 0.6, top-p 0.95, top-k 20, `min_p=0`, presence penalty 0, `enable_thinking` recorded in the serving metadata, a 2,048-token output budget, and a 128-token thinking budget. Metadata records prompt hashes, model identifiers, decoding settings, raw outputs, cleaned outputs, and job/runtime information for all lanes. Table 4 gives the decoding settings.

Lane	Model	Temp.	top-p	top-k	Pres.	Max toks	Mode
GPT-OSS low	`gpt-oss-20b`	0.7	0.95	none	0.0	3072	low
GPT-OSS medium	`gpt-oss-20b`	0.7	0.95	none	0.0	3072	medium
LLM-Structured	`gpt-oss-20b`	0.7	0.95	none	0.0	3072	medium + JSON
Qwen non-thinking	`Qwen3.6-35B-A3B-FP8`	0.7	0.8	20	0.0	512	non-thinking
Qwen thinking	`Qwen3.6-35B-A3B-FP8`	0.6	0.95	20	0.0	2048 + think 128	thinking

Table 4: LLM decoding settings recorded in the five-seed evaluation. LLM-Structured keeps the GPT-OSS medium decoding settings and changes only the output contract.

C.1 Prompt contract

Each one-call prompt-only LLM lane receives five groups of information. The **task instruction** asks for one English sentence and no explanation. The **frame fields** provide the scenario, target label, frame name, frame description, required roles, and optional slots. The **control fields** give clause type, illocutional profile, request strategy, modality, voice, polarity, valency, information packaging, and the applicability mask. The **negative constraints** forbid unsupported events, participants, slots, diagnoses, names, locations, prompt metadata, labels, and task descriptions. The **provenance fields** record prompt hash, model id, decoding settings, raw output, cleaned output, and runtime metadata.

C.2 One-call structured-output LLM ablation

The appendix-only LLM-Structured lane tests whether the one-call prompt-only LLM failures are mainly caused by weak output formatting. It uses GPT-OSS-20B medium with the same decoding settings as the sentence-only GPT-OSS medium lane, but changes the output contract. The model must return one JSON object with a `sentence` field, a `frame_commitment` object, and a `control_commitment` object. The comparison preserves the same one-call setting as the main LLM lanes: one call per plan, one candidate, no few-shot examples, no retry loop, no validator during generation, no n-best selection, no constrained decoding, and no post-hoc repair.

Only the `sentence` field is scored. The self-declared frame and control commitments are stored as diagnostics, but they are not used to select, repair, or credit the output. If the model does not return a parseable `sentence` field, the row is recorded as a structured-output failure and remains in the scored total. The same automatic scorer then evaluates the extracted sentence against the stored frame/control record. Table 5 reports the result.

Condition	Calls /plan	Output contract	Exact cfg. % ↑	Frame % ↑	Unsupported % ↓	Malformed % ↓	s ↓
GPT med.	1	sentence only	39.2±2.4	**96.8±1.4**	**0.0±0.0**	**0.0±0.0**	441.10±73.15
LLM-Struct.	1	JSON + one sentence	37.4±1.9	96.2±1.5	**0.0±0.0**	1.0±0.6	941.87±78.52
TraceGen	1 grammar pass	sentence + trace	**91.9±0.0**	91.9±0.0	**0.0±0.0**	8.1±0.0	**0.02±0.00**

Table 5: One-call structured-output LLM ablation. Values are mean ± standard deviation over five seeds. LLM-Structured uses GPT-OSS-20B medium with the same decoding settings as the sentence-only GPT-OSS medium lane, but requires one JSON object and scores only the extracted `sentence` field. The frame and control commitments are logged as diagnostics only.

The structured-output contract does not improve exact configuration over the sentence-only GPT-OSS medium lane. LLM-Structured obtains 37.4±1.9% exact configuration, compared with 39.2±2.4% for the sentence-only prompt. Frame preservation remains close, 96.2±1.5% versus 96.8±1.4%, but the JSON contract introduces 1.0±0.6% malformed rows when the model spends the full token budget in reasoning and emits no parseable `sentence` field. This suggests that the LLM control errors in this setting are not mainly an output-formatting artefact. The ablation tests output formatting only; it does not test a stronger generate–check–repair pipeline.

D Template Baseline Details

The template baseline uses fixed, reusable template families rather than learned generation. The reported comparison uses Template-5 and Template-10. For each evaluation plan, the template generator selects a reusable template whose declared control support is compatible with the requested configuration, fills its slots from curated inventories, records the template id and slot surfaces, and emits either one sentence or an explicit coverage

failure. Template-20 is not part of the seven-lane comparison.

The template baseline is treated as a released template record, not as an optimised control-indexed grammar. The appendix therefore reports the number of reusable families, their supported controls, coverage failures, and authoring-effort status where measured. Authoring time was not logged, so the paper reports `not_logged` rather than inferring it. The 14 template records were reused for 320 generated rows: 160 Template-5 rows and 160 Template-10 rows. No item-specific templates were authored; each family is indexed by compatible control requirements and filled from semantic-frame slots. The machine-readable inventory is stored in `template_control_inventory.json` and `template_effort.csv`.

D.1 Template slot maps

The template bank is filled from scenario-specific slots rather than from item-specific sentences. For SST-2, slots include evaluator or experiencer, target, sentiment predicate, copula, hedge, and booster. For HuRIC, slots include action verb, object or landmark, path/site preposition, manner, politeness marker, and optional vocative. For Symptom2Disease, slots include patient reference, symptom terms, duration, onset, severity, and body site. For SNIPS, slots include user/request phrasing and intent-specific values such as place, city, time, destination, cuisine, road, or recipient. The supplementary artefact stores the exact slot names and machine-readable maps.

D.2 Template inventory

The paper-facing template inventory contains 14 reusable template records grouped into 13 construction families. Each family record states its supported lanes, scenario support, required controls, compatible controls, required slots, example patterns, and manual-effort status. For example, a HuRIC directive family supports Template-5 and Template-10, requires a directive profile and transitive valency, can realise imperative or closed-interrogative clauses with direct, polite, or *could*-modal request strategies, requires predicate and patient slots, and uses patterns such as `{verb} {patient}.`, `Please {verb} {patient}.`, and `Could you {verb} {patient}?`. Manual authoring time is recorded as `not_logged`; the paper therefore reports template count, family count, reuse count, and coverage failures instead.

These details make the baseline auditable without presenting the full template bank in the paper. The complete machine-readable template records are included in the supplementary artefact.

E Control Inventory Details

This appendix defines the full frame/control grid used in the evaluation. The source files are `semantic_frames.json`, `control_inventory.json`, and `control_grid_160.jsonl`. Together they define the 32 semantic-frame prototypes, the wording-control options, the 160 plan rows, the applicability masks, and the forbidden-content rules. The same requested-control object and applicability mask are passed to all seven primary lanes. Across five seeds, the primary audit finds all 5,600 expected records and retains 65 explicit TraceGen coverage failures.

E.1 Source files and plan schema

The `control_inventory.json` file defines the visible wording controls scored in exact-configuration and per-control metrics. The `semantic_frames.json` file stores the 32 frame prototypes, including frame id, scenario, frame type, label, predicate lemma, participants, slots, default valency, and forbidden additions. The `control_grid_160.jsonl` file contains the exhaustive 160-row evaluation set. Each row combines one semantic frame with one requested-control object, an applicability mask, a profile index, a source hash, and row-level forbidden-content constraints. The generated-output records then add the raw and cleaned output, system lane, seed, prompt or source record, trace or template id where available, scorer flags, and explicit failure records.

Each plan row is traceable without exposing raw JSONL in the paper. The identifiers and provenance fields link each output back to a unique pre-generation row. The scenario and semantic-frame id identify the source family and prototype. The semantic-frame object states what must remain true in a valid output: event or intent, required role fillers, required slots or modifiers, and lexical predicate family. The requested-controls object states how that frame should be worded, and the applicable-controls list states which controls enter the denominator for that row. The forbidden-

content field records constraints such as no extra event, no unlisted participant or slot, and no metadata leakage.

E.2 Semantic-frame prototypes

The 32 prototypes are distributed evenly across the four scenario families before expansion into the 160 plans. HuRIC contributes robot-command frames for manipulation, navigation, perception, delivery, placement, device control, and communication, with roles such as agent, patient, and recipient, and slots such as destination, location, recipient, and source. SNIPS contributes intent-query frames for weather, restaurant booking, place search, directions, ETA sharing, place comparison, traffic, and place details, with slots such as location, destination, party size, route, source, and time. SST-2 contributes positive and negative sentiment/evaluation frames with reviewer and patient roles plus an aspect slot. Symptom2Disease contributes symptom-report frames with experiencer and symptom roles plus body-site, duration, severity, time, and trigger slots.

E.3 Control inventory

The grid scores eight wording-control dimensions. Clause type distinguishes declarative, imperative, closed interrogative, open interrogative, and exclamative forms. Illocutional profile distinguishes assertive, directive, and expressive constructions as grammar-visible profiles rather than claims about full social force. Request strategy distinguishes no request, direct imperative, polite directive, ability-modal interrogative, willingness-modal interrogative, *mind* interrogative, and formal passive request. Modality records *can*, *could*, *would*, *should*, or no modal. Voice records active and passive mappings, including by-agent and retained-object passive variants where licensed. Polarity distinguishes affirmative and negative realisations. Valency records intransitive, transitive, double-object, recipient-PP, PP-complement, complex-transitive, copular, clausal-complement, and particle-verb frames. Information packaging records neutral order, vocative, fronted time, fronted location, and supplement choices.

E.4 Applicability mask and scenario allocation

Each row stores `requested_controls` and `applicable_controls`. Requested controls are fixed before generation, whereas realised controls are inferred after generation. A system may fail to realise an applicable control, but it may not silently replace the requested control with an easier one. If the requested configuration is outside a system's coverage, the row is retained as an explicit coverage failure.

The 160-plan set contains 48 HuRIC rows, 40 SNIPS rows, 40 SST-2 rows, and 32 Symptom2Disease rows. HuRIC stresses directive form, request strategy, modality, voice, polarity, valency, and vocatives. SNIPS stresses query/request form, slot preservation, interrogatives where licensed, fronted time/location, and supplements. SST-2 stresses assertive versus expressive profiles, polarity, clause type, copular and complex-transitive valency, and supplements. Symptom2Disease stresses declarative symptom reports, severity and duration modifiers, and checks against unsupported clinical content.

E.5 Reader-facing examples

A HuRIC request row may specify an OPEN frame with actor=addressee and patient=door, forbid extra actions or unrequested locations, and request a closed-interrogative directive with *could*, active voice, affirmative polarity, transitive valency, and neutral packaging. *Could you open the door?* satisfies that row. A SNIPS query row may specify a weather-query event with requester=speaker, location=Boston, and time=tomorrow, forbid an unrequested location or extra event, and request fronted time. *Tomorrow, please check the weather in Boston.* satisfies that row. The machine-readable file uses the same fields for all 160 plans.

E.6 Automatic scoring contract

The automatic metrics follow the contract defined in the main text: a successful controlled output preserves the frame, realises the requested controls, and leaves an auditable generation record. The denominator is the number of generated records counted for a percentage. In the headline automatic table, this is 160 records for one system lane and one seed, or 800 records for one lane across five seeds. Explicit coverage failures, empty outputs, and malformed outputs stay in the denominator.

The main paper reports exact configuration and frame preservation as headline automatic scores, then summarises generation success and coverage failures in prose. This appendix groups the full diagnostic bundle by the three parts of the contract. Repetition and runtime are

practical diagnostics rather than success conditions. Distinct-1 and distinct-2 follow Li et al. (2016); Self-BLEU is adapted from BLEU (Papineni et al., 2002) and from its use as a diversity diagnostic (Zhu et al., 2018). Table 6 gives the paper-facing metric definitions used to generate `automatic_scored_outputs_1120.jsonl` and the appendix tables.

E.7 Checker validity and failure accounting

The automatic checker is a reproducible diagnostic, not an oracle for meaning. It uses the requested controls and applicability mask, then applies deterministic tests to the cleaned generated sentence. It does not use a TraceGen grammar trace to mark a surface control or frame as correct. For templates, the template id and slot map are source records; for one-call prompt-only LLM lanes, the prompt and output record are source records; for TraceGen, rule traces are reported separately as derivational evidence. Table 7 summarises the safeguards and known limitations.

The primary seven-lane five-seed audit has 5,600/5,600 expected records, no missing keys, no duplicate keys, no frame/control mismatches, and 65 explicit TraceGen coverage failures retained in the total scored rows. Adding the appendix-only LLM-Structured lane contributes 800 records. The combined artefact audit has 6,400/6,400 expected records and 73 explicit failure or empty-output rows: 65 TraceGen coverage failures plus 8 LLM-Structured rows where the model emitted no parseable `sentence` field. These are grammar-coverage or one-call structured-output failures rather than silent substitutions, and they remain in the scored totals. Remaining one-call prompt-only LLM frame failures in the scored JSONL files are retained with component flags, such as dropped locations in HuRIC, dropped aspects in SST-2, or intent changes in SNIPS. The scored records preserve these examples so that the verification audit can distinguish semantic errors from automatic-checker boundary cases.

F Experimental Configuration Details

This appendix records the execution contract for the reported evaluation. The combined outputs, automatic metrics, validation reports, and generated result tables are stored with the evaluation artefact.

F.1 Datasets and sampling contract

The evaluation uses one shared 160-plan set and seven primary system lanes. Table 8 gives the counts needed to reproduce the denominators of the reported metrics.

Item	Value
Scenario families	SST-2, HuRIC, Symptom2Disease, and SNIPS.
Evaluation set	160 semantic-frame/control plans.
Scenario allocation	HuRIC 48, SNIPS 40, SST-2 40, Symptom2Disease 32.
Primary lanes	Template-5, Template-10, TraceGen, GPT-OSS low, GPT-OSS medium, Qwen non-thinking, Qwen thinking.
Generated records	160 plans crossed with seven lanes, per seed; 5,600 records across five seeds.
Human audit	24 plans crossed with seven lanes, giving 168 blinded outputs.
TraceGen capability suite	80 appendix-only rows, not part of the all-method comparison.

Table 8: Execution contract for the reported evaluation. Explicit failures remain in these counts.

F.2 Generated files and validation checks

The evaluation artefact contains five groups of files: the control inventory and 160-plan list; one output file per system lane; combined and automatically scored JSONL outputs; metric CSVs for control accuracy, frame preservation, malformed output, unsupported content, repetition, runtime, traces, and slot surfaces; and template-effort, validation-summary, and manifest files. Exact filenames are stored in `MANIFEST.json`. The paper tables are generated from the same artefact. Table 9 lists the validation checks applied before paper numbers are inserted.

F.3 Execution and compute metadata

Runtime rows are reported in `runtime_by_condition.csv` and Table 13. Token accounting is not reported uniformly across all lanes, so the paper reports wall time and rows per second rather than normalised token throughput. The primary one-call prompt-only LLM lanes were executed as 20 single-GPU jobs on NVIDIA A100-PCIE-40GB devices on a Linux cluster, using vLLM 0.21.0, Python 3.12.12, CUDA 13.0, and NVIDIA driver 580.159.03. GPT-OSS-20B has 20.9B total parameters and 3.6B active parameters. Qwen3.6-35B-A3B-FP8 has 35B total parameters and 3B activated parameters. The four primary LLM lanes required 11,177 seconds of summed generation time, approximately 3.1 single-GPU hours. The appendix-only

Requirement	Metrics	Denominator	How it is measured
Meaning preserved	Generation success; frame preservation; unsupported-content flags; malformed-output flags; Symptom2Disease severity handling.	All 160 plans for one system lane and one seed. Explicit coverage failures, empty outputs, and malformed outputs stay in this total and score as failures where relevant.	The output must be one usable sentence and preserve the intended event, required participants, required slots, and event count. Unsupported additions, metadata leakage, extra events, and unsupported clinical claims are recorded as failure diagnostics. Licensed predicate alternants are accepted when the required frame material remains present.
Requested controls realised	Exact configuration; per-control surface accuracy.	Exact configuration uses all 160 plans for one system lane and one seed. Per-control accuracy uses only records where that control is applicable; for example, modality is not counted when no modal choice is meaningful for the plan.	The scorer infers realised controls from the output surface and compares them with the requested controls. Exact configuration is all-or-nothing: the record is correct only if every applicable requested control matches. Per-control accuracy reports the same comparison separately for clause type, illocutional profile, request strategy, modality, voice, polarity, valency, and information packaging.
Generation auditable	Source-record availability; TraceGen trace availability; trace-configuration accuracy for TraceGen; trace diversity and mean trace length.	Source and trace availability use all 160 plans for one system lane and one seed. Trace diversity and trace length use only records that actually have grammar traces, with trace availability showing how much coverage was achieved.	Templates provide a template id and slot map, one-call prompt-only LLM lanes provide prompt and decoding records, and TraceGen provides grammar traces for successful grammar outputs. Trace availability is evidence for provenance, not extra credit for surface-control or frame preservation. Coverage failures remain in the total scored rows and are recorded explicitly.

Table 6: Automatic metric definitions grouped by the three requirements a controlled output should meet. The denominator column states what is counted as the total for each percentage. Repetition and runtime are reported separately as practical diagnostics, not as success conditions. These definitions are implemented by the released table-generation script.

Issue	How it is handled	Reason
Requested versus realised controls	Both are stored. Exact configuration is true only when all applicable requested controls are recovered from the surface.	Prevents a generator from silently replacing a hard request with an easier one.
Frame semantics versus lexical form	The frame check accepts licensed semantic alternants, but only when required frame material is still present.	Avoids cherry-picking away paraphrases while still penalising dropped roles, dropped slots, event changes, and unsupported additions.
Trace versus surface score	Trace availability and trace-configuration accuracy are separate from surface-control accuracy.	Avoids giving TraceGen credit merely because its internal trace records the requested feature.
Repairs and coverage	Repaired or unsupported configurations remain in the total scored rows and are flagged in validation files.	Makes the table a compatibility result over the pre-specified plans rather than a post-hoc success-only sample.
Human verification	The audit pack asks annotators to judge frame preservation, requested form, unsupported additions, malformed output, naturalness, and rigidity from the frame/control record and sentence.	Separates reproducible automatic diagnostics from adjudicated human verification.

Table 7: Automatic-checker safeguards and known limitations.

Validation group	Check before inserting paper numbers
Plan identity	Every row preserves the plan id, semantic frame, requested controls, and applicability mask.
Lane coverage	Every plan has exactly seven primary-lane records or explicit failure records.
Failure accounting	Coverage failures, empty outputs, and malformed outputs remain in the scored totals.
Control accounting	Requested controls, realised controls, and applicability masks are stored separately.
Source records	Templates keep template family and slot surfaces; TraceGen keeps grammar features and traces; one-call prompt-only LLM lanes keep prompt hash, raw output, cleaned output, model id, and decoding settings.
Metric provenance	Automatic metric files and validation reports are generated from the same artefact used for the paper tables.
Final audit	The seven-lane five-seed audit contains all 5,600 expected records and no missing keys, duplicate keys, or frame/control mismatches.

Table 9: Validation checks used to guard the evaluation artefact. The detailed manifest is released with the artefact.

LLM-Structured diagnostic adds 4,709 seconds, approximately 1.3 single-GPU hours. Jobs were scheduled in parallel, so elapsed wall-clock span is not the same as summed GPU time.

F.4 Optional external-adapter execution audit

The supplementary artefact includes an appendix-only execution audit for SemRealizer adapters to SimpleNLG, ERG/MRS through ACE (Packard, 2013), and OpenCCG. These rows are not part of the seven-lane comparison and are not quality baselines. They record whether each adapter produced a record or structured failure under the same 160-plan, five-seed evaluation set. SimpleNLG uses a manual SVO-style bridge. ERG/MRS uses a reference-roundtrip check from frame-derived grammatical text rather than direct frame-to-MRS generation. OpenCCG requires a grammar-specific HLDS/XML logical-form bridge. The released adapter report records the frame-format mapping account: SimpleNLG bridge JSON is executable; supplied SimpleMRS or reference-roundtrip input is executable for ERG/ACE; and frame-derived OpenCCG role strings are marked as non-executable candidates unless a plan supplies a grammar-licensed logical form. This audit is included for reproducibility and interface documentation only. It is not used to support the automatic or human comparison.

G Automatic Metrics

The automatic numeric tables in this appendix are generated from the evaluation artefact. Values are computed programmatically from seeds 0–4 and shown as mean ± standard deviation unless the table is a static resource inventory. Metric headers use ↑ for higher-is-better and ↓ for lower-is-better; **bold** marks the best displayed value in a comparable result column and underlining marks the second-best distinct displayed value. Static inventories and single-row accounting tables are left unranked. Tables 10, 11, 12, and 13 report frame diagnostics, per-control accuracy, trace and slot diagnostics, and diversity/runtime diagnostics.

Template effort and coverage accounting are compact enough to state directly. The template bank contains 14 reusable records in 13 construction families, reused across 320 generated rows for Template-5 and Template-10. No item-specific templates were authored, no template coverage failures were recorded, and authoring time was not logged.

Condition	Pred. % ↑	Part. % ↑	Slots % ↑	One event % ↑	No meta % ↑	Single sent. % ↑	No clin. add. % ↑
Tpl-5	**100.0±0.0**	**100.0±0.0**	**100.0±0.0**	**100.0±0.0**	**100.0±0.0**	**100.0±0.0**	**100.0±0.0**
Tpl-10	**100.0±0.0**	**100.0±0.0**	**100.0±0.0**	**100.0±0.0**	**100.0±0.0**	**100.0±0.0**	**100.0±0.0**
TraceGen	91.9±0.0	91.9±0.0	91.9±0.0	91.9±0.0	**100.0±0.0**	**100.0±0.0**	**100.0±0.0**
GPT low	99.8±0.3	99.9±0.3	96.5±2.1	**100.0±0.0**	**100.0±0.0**	**100.0±0.0**	**100.0±0.0**
GPT med.	99.9±0.3	99.5±0.3	97.0±1.5	**100.0±0.0**	**100.0±0.0**	**100.0±0.0**	**100.0±0.0**
Qwen NT	99.1±0.7	99.9±0.3	90.3±2.0	**100.0±0.0**	**100.0±0.0**	**100.0±0.0**	**100.0±0.0**
Qwen T	96.0±1.1	98.1±0.0	91.9±1.4	**100.0±0.0**	99.9±0.3	**100.0±0.0**	**100.0±0.0**

Table 10: Automatic frame-component and unsupported-content diagnostics. Values are mean ± standard deviation over five seeds. Pred., Part., Slots, and One event are the components of the frame score. No meta, Single sent., and No clin. add. are adverse automatic checks reported as pass rates.

For TraceGen, generation success, trace availability, and well-formed output are all 91.9±0.0%, while explicit coverage failures are 8.1±0.0%.

Trace diagnostics are computed over the five seeded runs. TraceGen has 147 traced rows per seed out of 160 attempted rows, 52 unique rule traces per seed, rule entropy 5.23, and mean trace length 5.01 rules. Across all five seeds, this gives 735 traced rows out of 800 attempts; the remaining 65 rows are explicit coverage failures. Slot-surface diagnostics are similar across lanes: most conditions use 81 unique slot-surface types per seed, while TraceGen uses 76 because failed coverage rows do not contribute slot surfaces.

Condition	Clause % ↑	Illoc. % ↑	Req. % ↑	Modal % ↑	Voice % ↑	Pol. % ↑	Val. % ↑	Pack. % ↑
Tpl-5	**100.0±0.0**	**100.0±0.0**	72.7±0.0	**100.0±0.0**	93.1±0.0	**100.0±0.0**	84.4±0.0	**95.6±0.0**
Tpl-10	90.0±0.0	**100.0±0.0**	72.7±0.0	**100.0±0.0**	93.1±0.0	**100.0±0.0**	84.4±0.0	**95.6±0.0**
TraceGen	91.9±0.0	91.9±0.0	**85.2±0.0**	85.2±0.0	91.9±0.0	91.9±0.0	**91.9±0.0**	91.9±0.0
GPT low	87.0±1.4	**100.0±0.0**	52.3±1.8	93.0±3.0	96.6±0.7	99.0±0.3	74.9±1.0	74.0±2.8
GPT med.	90.8±0.8	**100.0±0.0**	56.4±4.0	96.1±2.2	**97.5±0.8**	98.5±1.6	73.6±0.7	78.2±1.7
Qwen NT	93.9±0.3	**100.0±0.0**	84.8±2.5	95.9±2.1	97.1±0.8	99.4±0.4	72.6±0.7	60.5±1.5
Qwen T	96.9±1.2	**100.0±0.0**	71.6±5.3	97.7±2.3	**97.5±0.4**	98.8±1.0	78.0±1.3	77.6±2.2

Table 11: Per-control automatic surface accuracy for applicable requested controls in the evaluation. Values are mean ± standard deviation over five seeds. Illoc. is illocutional-profile accuracy, Req. is request-strategy accuracy, and Pack. is information-packaging accuracy.

Condition	Source % ↑	Trace % ↑	Unique slots ↑	Slot entropy ↑	Unique traces ↑	Trace entropy ↑	Trace len. ↑
Tpl-5	**100.0±0.0**	0.0±0.0	**81.0±0.0**	**5.89±0.00**	0.0±0.0	0.00±0.00	0.00±0.00
Tpl-10	**100.0±0.0**	0.0±0.0	**81.0±0.0**	**5.89±0.00**	0.0±0.0	0.00±0.00	0.00±0.00
TraceGen	91.9±0.0	**91.9±0.0**	76.0±0.0	5.82±0.00	**52.0±0.0**	**5.23±0.00**	**5.01±0.00**
GPT low	**100.0±0.0**	0.0±0.0	**81.0±0.0**	**5.89±0.00**	0.0±0.0	0.00±0.00	0.00±0.00
GPT med.	**100.0±0.0**	0.0±0.0	**81.0±0.0**	**5.89±0.00**	0.0±0.0	0.00±0.00	0.00±0.00
Qwen NT	**100.0±0.0**	0.0±0.0	**81.0±0.0**	**5.89±0.00**	0.0±0.0	0.00±0.00	0.00±0.00
Qwen T	**100.0±0.0**	0.0±0.0	**81.0±0.0**	**5.89±0.00**	0.0±0.0	0.00±0.00	0.00±0.00

Table 12: Source-record, grammar-trace, slot-surface, and trace-diversity diagnostics. Values are mean ± standard deviation over five seeds. Template and one-call prompt-only LLM rows have source records but no grammar derivation traces; TraceGen trace values are computed on successful grammar outputs, with coverage failures retained in trace availability.

Condition	Unique % ↑	Dup. % ↓	D-1 ↑	D-2 ↑	Self-BLEU proxy ↓	s ↓	rows /s ↑
Tpl-5	**100.0±0.0**	**0.0±0.0**	0.120±0.000	0.279±0.000	0.987±0.000	**0.01±0.00**	22218.10±276.99
Tpl-10	**100.0±0.0**	**0.0±0.0**	0.119±0.000	0.277±0.000	0.987±0.000	**0.01±0.00**	**22975.35±217.30**
TraceGen	**100.0±0.0**	**0.0±0.0**	0.138±0.000	0.315±0.000	0.975±0.000	0.02±0.00	10507.42±1055.81
GPT low	94.9±0.9	5.1±0.9	0.142±0.001	0.326±0.005	0.974±0.002	190.74±84.89	0.93±0.26
GPT med.	95.6±0.6	4.4±0.6	0.144±0.003	0.342±0.005	0.971±0.005	441.10±73.15	0.37±0.06
Qwen NT	95.0±1.6	5.0±1.6	0.147±0.004	0.329±0.009	0.971±0.003	179.33±89.09	1.01±0.30
Qwen T	98.8±0.4	1.2±0.4	**0.168±0.008**	**0.402±0.011**	**0.953±0.005**	1424.27±156.39	0.11±0.01

Table 13: Repetition and runtime diagnostics for the evaluation. Values are mean ± standard deviation over five seeds. The Self-BLEU value is a deterministic unigram-overlap proxy used for bundle-level comparison, not a learned quality metric.

H Explicit Control and Human Audit Details

The explicit audit uses the same semantic-frame/control plan identifiers across generated lanes. Validation flags are retained for rows whose logged controls differ after generator-side repair. Requested linguistic configurations are shown for every evaluation row, together with an applicability mask for controls that are not semantically licensed by a plan. The audit is therefore a two-annotator contract-verification check over a matched subset.

H.1 Audit design

The audit begins with a 24-item pilot, which is excluded from final analysis. The audit is internal because both annotators are paper authors. It is presented as contract verification and disagreement diagnosis, not as an independent estimate of user preference or deployment quality. The final audit pack contains 192 generated outputs: 24 semantic-frame/control items crossed with the seven main lanes plus the appendix-only LLM-Structured condition. Both annotators completed the first pass for all 192 outputs, giving 384 judgement rows and 192 double-coded items. The final export contains 192 consensus rows: 86 explicit adjudications for first-pass disagreements and 106 rows where the two first-pass labels already matched. Within that export, the main seven-lane audit contributes 168 consensus outputs, while LLM-Structured contributes 24 appendix-only outputs. One GPT-OSS medium item is marked unresolved and counted as not manually faithful in the strict primary rate.

The 24 plan-control items are selected from the pre-specified 160-plan evaluation set before system-lane labels are shown to annotators. The sampling strata cover scenario family, request strategy, voice, polarity, valency, and information packaging. Explicit generation failures remain in the matched sample if their plan is selected. The design is matched for coverage and estimation precision, not for null-hypothesis testing. In the seven-lane main audit, $n = 168$ consensus rows give a worst-case Wilson 95% interval half-width of about 7.4 percentage points at $p = 0.5$; per-lane consensus rates have $n = 24$ and are therefore descriptive (Wilson, 1927).

For reproducibility, the supplementary material provides the annotation web UI in `annotationUI/`, including the blinded task pack, system-lane key, manifest, blinding report, annotation templates, static assets, and export/adjudication endpoints used for the two-annotator pass. The audit follows the separation between first-pass reliability and adjudicated labels recommended for computational-linguistics annotation (Artstein and Poesio, 2008). Because several error labels are rare, the released audit export records prevalence, confusion counts, and positive/negative agreement rather than relying on Cohen's κ alone (Cohen, 1960; Byrt et al., 1993; Cicchetti and Feinstein, 1990).

H.2 Annotator status, consent, and payment

The verification audit was carried out by the two paper authors. No crowdworkers, students, or external annotators were recruited or paid. The annotators judged generated sentences against synthetic frame/control records using the audit interface and codebook described here. Because the annotators were the authors and no external human-subject data were collected, no participant recruitment or payment protocol was used. The audit involved generated outputs and synthetic frame/control records only, and did not collect personal data from human participants. No ethics-board review was obtained because the audit used author annotators and generated/synthetic records only; this status is reported in the Responsible NLP checklist.

Adjudication is by consensus after the independent first pass. The adjudication sheet preserves both first-pass labels, exposes disagreements to the annotators, records a final adjudicated label and a short note, and permits an unresolved outcome only when consensus cannot be reached.

H.3 Audit metrics and denominators

The audit starts from the semantic frame, requested controls, and generated sentence. Annotators then answer concrete questions about the sentence. The primary label, **manual faithful**, asks whether the whole frame/control contract is satisfied after adjudication. Component labels check whether the frame is preserved, whether the requested form is realised, whether the request strategy is correct for directive/request items, whether the output has no unsupported content, and whether it is a single usable sentence. Diagnostic labels record naturalness and rigid wording; these help interpret outputs but do not enter the primary faithfulness score.

The denominator for each primary system lane is 24 consensus outputs. First-pass agreement is reported separately over paired judgement labels. If a

question is not applicable, for example request strategy on a non-request item, that item is excluded only from that component metric and the applicable count is recorded. The main seven-lane audit contains 168 outputs. The full exported audit pack contains 192 outputs because it also includes the appendix-only LLM-Structured condition.

H.4 Example audit cards

For `ssttwo_030`, the card shows an SST-2 negative-review frame: the predicate is *reject*, the reviewer is *the reviewer*, the patient is *the ending*, and the aspect slot is *resolution*. The requested controls are exclamative clause type, expressive profile, supplement packaging, no modality, affirmative polarity, no request strategy, transitive valency, and active voice. The applicable controls are clause type, illocutional profile, voice, polarity, valency, and information packaging. Annotators judge requested-form questions only against that mask: a dropped aspect or ordinary declarative form is a failure, while request strategy is not scored for this non-request item.

For `huric_041`, the card shows a HuRIC device-control frame: the robot should turn on *the floor lamp*, and the location slot is *bedroom*. The requested controls are closed interrogative clause type, directive profile, neutral packaging, *would* modality, negative polarity, *mind* interrogative request strategy, particle-verb valency, and active voice. The applicability mask includes all eight controls. *Would you mind not turning on the floor lamp in the bedroom?* preserves the patient, location, negation, particle verb, and request strategy. *Would you not turn on the floor lamp in the bedroom?* keeps the frame but fails the requested *mind* strategy.

H.5 Audit export schema

The annotation server exports the human-audit data in three layers: first-pass annotations, consensus labels, and summary rates. The paper-facing human table is generated from the complete consensus export rather than entered by hand. If consensus cannot be reached, the item is marked unresolved, counted as not manually faithful in the strict primary rate, and reported separately in the export. The rebuild files are `first_pass_annotations_joined.csv` for independent labels and notes, `adjudicated_annotations_joined.csv` for one consensus row per audited output, `agreement_by_label.csv` for reliability diagnostics, `explicit_adjudications_joined.csv` for disagreement-resolution records, and `adjudicated_rates_by_group.csv` for final lane-level rates and Wilson intervals.

Table 14 reports first-pass agreement by label for the complete double-coded audit.

H.6 Verification-audit codebook

The audit pack and codebook are included so that the automatic results can be checked independently. The annotation interface records eleven labels. Q1 checks frame preservation. Q2 checks required participants, entities, slots, and modifiers. Q3 checks requested linguistic form, including clause type, voice, tense, polarity, modality, valency, and information packaging. Q4 checks request strategy when that control is applicable. The reversed primary labels record Q5 unsupported extra events, Q6 unsupported slots or participants, Q7 topic words preserved with the event changed, Q8 metadata leakage, and Q9 malformed or unusable output. Q10 records whether the sentence is natural enough for the intended genre, and Q11 records rigid or boilerplate wording. Q3 failure types are clause type, voice, tense, polarity, modality, request form, valency, information packaging, and other. For Q3 and Q4, annotators judge only controls shown as applicable in the audit panel; for example, an SST-2 expressive exclamative with `request_strategy=none` is not penalised for request strategy because request strategy is not applicable.

The consensus `manual_faithful` label is true when Q1 is Yes; Q2, Q3, and Q4 are not No; and Q5–Q9 are No. Unresolved rows are counted as not manually faithful in the strict primary rate and reported in the manifest. Q10 naturalness and Q11 rigidity are diagnostic and are not included in the primary faithfulness score. The audit export schema records, for each label, the applicable item count, first-pass percent agreement, Cohen's κ where prevalence permits, positive and negative agreement, 2-by-2 confusion counts, the consensus rate, and a Wilson 95% confidence interval.

H.7 Failure typology

The audit distinguishes six recurrent error families: wrong requested form, topic words preserved with the event changed, unsupported event addition, unsupported slot or participant addition, metadata leakage, and malformed or unusable output. These

Label	N	Agree ↑	κ ↑	Pos. ↑	Neg. ↑	Main disagreement pattern
Q1 frame	192	99.5	0.89	99.7	88.9	1 frame disagreements
Q2 participants/slots	192	99.5	0.95	99.7	95.7	1 preservation disagreements
Q3 requested form	192	83.3	0.65	86.6	78.1	32 Yes/No form disagreements
Q4 request strategy	192	96.9	0.95	96.2	97.3	6 request-strategy disagreements
Q5 extra event	192	98.4	-0.01	0.0	99.2	3 extra-event disagreements
Q6 extra slot/participant	192	99.0	0.83	83.3	99.5	2 extra-slot/participant disagreements
Q7 event changed	192	99.0	0.91	91.7	99.4	2 event-change disagreements
Q8 metadata leak	192	100.0	–	–	100.0	No disagreements
Q9 unusable	192	93.2	0.54	58.1	96.3	13 malformed/usability disagreements
Q10 natural enough	192	78.6	0.42	85.9	55.9	41 naturalness disagreements
Q11 rigid wording	192	88.5	0.21	26.7	93.8	22 rigidity disagreements

Table 14: First-pass agreement by audit label for the complete double-coded audit. Agreement is reliability evidence; final human verification rates are generated from consensus labels rather than first-pass rates.

labels are intentionally more concrete than a single fluency score. A fluent sentence can still be wrong for this task if it changes the frame, adds an unlicensed symptom or location, or satisfies a different request strategy from the requested one. In the running OPEN-the-door frame, *Could you open the door?* passes because it keeps the event, addressee, door, and modal-interrogative form. By contrast, *Please open the door.* preserves much of the frame but uses the wrong request strategy; *Could you open the door and turn on the light?* adds an unsupported event; *Could you open the window?* changes the required patient; *[no sentence: requested combination not supported]* is an explicit coverage rejection; *The target label is GetWeather.* leaks metadata; and *[no parseable sentence]* remains in the scored set because no usable sentence can be checked.

I Additional Request-Form Examples

Runtime for the reported evaluation is reported in Table 13; the artefact also includes per-condition runtime CSV files and raw generation metadata. This appendix gives additional request-form examples used to interpret the grammar. TraceGen records store the corresponding frame, configuration, and selected grammar choices for each generated sentence.

A direct-imperative request accepts *Open the door.* and rejects *Could you open the door?*. A modal-interrogative request does the reverse. A formulaic indirect request accepts *Would you mind opening the door?*. A vocative-plus-modal request accepts *Pat, could you open the door?*. Finally, the one-event constraint accepts *Could you open the door?* and rejects *Could you open the door and turn on the light?*, even though the rejected sentence is grammatical English.

J Released Software and Additional Details

The anonymised supplementary artefact contains the review-time implementation, experiment configuration, prompts, frame/control files, generated outputs, scoring scripts, audit pack, trace examples, environment specifications, and the annotation web UI in `annotationUI/`. Upon acceptance, we will release a versioned public repository and PyPI package for `semrealiser`, covering the template, prompt-based, and TraceGen grammar-guided realisation interfaces used in this paper. The software release supports reproducibility and reuse of the frame/control contract; it is not claimed as a separate technical contribution.

J.1 Data statement and intended use

The released frame/control plans and generated outputs are synthetic English records derived from four cited scenario families. They are intended for reproducing the controlled-generation evaluation, inspecting frame/control failures, and testing annotation procedures. They are not records of real user language and should not be mixed with natural corpora without provenance labels. The Symptom2Disease examples are synthetic symptom reports for generation experiments; they are not clinical cases, diagnoses, or medical advice. The artefact keeps source scenario, plan id, semantic frame, requested controls, system lane, output, and validation metadata with each row. Redistribution should follow the licences and constraints of the source datasets and the supplementary artefact. Non-intended uses include clinical decision support, claims about real user behaviour, and training or evaluation settings where synthetic provenance is hidden.

J.2 Licences and release scope

We release the original artefacts created for this paper, including the frame/control grid, generated outputs, scoring scripts, audit schema, prompt templates, validation reports, and table-generation scripts, subject to the licences and access conditions of the source resources. Code is released under Apache 2.0. Paper-created data artefacts that we are permitted to redistribute are released under CC BY 4.0, with citation to this paper requested through the accompanying `CITATION.cff` file. Third-party datasets are not relicensed by this release. Where source terms restrict redistribution, we release only derived synthetic frame/control records and provenance metadata that can be shared under those terms.

Resource	Licence / terms and release treatment
HuRIC	Robot-command scenario family; Apache 2.0; cited, with derived frame/control records released.
SNIPS / Sonos SLU	Intent-query scenario family; research-only terms, no commercial use, and republication only unmodified under the same terms; cited, with no relicensing of source data and only permitted derived or synthetic records and provenance metadata shared.
SST-2 / GLUE	Sentiment/evaluation scenario family; source licence not clearly stated in the checked dataset cards; cited, with no original SST-2 text redistributed and only synthetic or abstracted frame records released where permitted.
Symptom2Disease	Symptom-report scenario family; CC0: Public Domain; cited, with synthetic derived records released under the medical non-use warnings above.
TraceGen artefact	Paper-created configuration, output, scoring, prompt, audit, and validation files; CC BY 4.0 for data and documentation, Apache 2.0 for code; released with provenance metadata, release-scope notes, and `CITATION.cff`.

Table 15: Source-resource terms and release treatment. Third-party datasets are cited but not relicensed by this paper.

J.3 Artefact checklist

The anonymous review artefact is organised around a simple review path: inspect one plan end to end and regenerate each automatic table. Reviewers can run the table-generation script from the evaluation artefact; verify that the primary comparison files contain 5,600 records across seven system lanes and five seeds, with the appendix-only LLM-Structured lane adding 800 records; inspect one plan across all primary lanes; regenerate Table 1 from the control and frame CSV files; regenerate trace-diversity diagnostics from the TraceGen trace file; inspect the blinded audit CSV and adjudication schema; and, if desired, launch the annotation web UI from `annotationUI/`.

The manifest records checksums, row counts, seed metadata, decoding settings, and validation gates. The supplementary artefact also includes the Python environment specification, package versions, model identifiers, decoding settings, prompt hashes, scorer scripts, table-generation scripts, validation reports, and checksums needed to rebuild the automatic tables and inspect the human-audit exports.

K Use of AI Assistants

The authors used AI assistants during the preparation of this work for writing assistance, code development and debugging, and suggestions of candidate lexical entities before human curation. AI assistants were not used as authors, annotators, or autonomous sources of evidence. All technical claims, citations, experimental results, code changes, lexical inventories, and final paper text were reviewed and verified by the authors. Candidate lexical entities suggested by AI tools were manually checked, edited, and either accepted or rejected before inclusion in the curated inventories. The final semantic frames, control grid, generated outputs, metrics, audit labels, and conclusions are the responsibility of the authors.