SENSA: UNSUPERVISED RULE GENERATION FOR MORE ROBUST TEXTUAL SUMMARIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM summarization/compaction has become an increasingly important use case of LLMs as context sizes increase along with token prices. We introduce SENSA (Summarize, Evaluate, Nudge, Summarize Again), a novel summarization method that uses lightweight domain-aware preprocessing to steer summarization into attending more strongly to important information that is commonly lost during the summarization process. We demonstrate that in most domains, summarization errors can be described by a low-dimensional set of categories, and rules based on these categories can be used to guide summarization within that domain to be more resilient against information loss while keeping the same tight summarization budget. We demonstrate SENSA on both technical and non-technical domains, and show it beats SOTA compression on information retention for a wide range of domains and practical compression budgets. SENSA performs especially well on technical fields like coding, where relative improvement in compression information retention can be as high as 100% compared to the next best SOTA method.

1 INTRODUCTION

As context windows grow in LLM applications, the need for robust algorithms to shorten LLM inputs has never been higher. Long contexts not only significantly increase costs commensurate with currently surging token prices, but also induce avoidable errors as needle-in-a-haystack information retrieval becomes more and more difficult with increased context lengths (Kamradt, 2023; Modarressi et al., 2025). These errors can pose critical issues as LLMs become more and more integrated with safety-critical systems, like those involved in healthcare (Singhal et al., 2023; Van Veen et al., 2024) or financial infrastructure (Wu et al., 2023; Li et al., 2023a).

Most textual summarization or compaction techniques take a domain-agnostic approach to the task (Jiang et al., 2023; Li et al., 2023b; Pan et al., 2024); regardless of the type of corpus, modern summarization algorithms are generally single-pass methods that take in a long context and output a shorter one. Through this process, there is no intrinsic awareness of whether we are trying to compress code, or creative writing, or a legal document. However, the challenges of compressing text in different domains is, predictably, quite variable.

One work that comes close to breaking this symmetry is Agent Context Optimization (ACON) (Kang et al., 2026a), which specifically uses errors during compression to guide the compression prompt of the main LLM performing the summarization. This work comes close to a reasonable solution by convincingly suggesting that errors on the specific input corpus can guide a compression system to a better optimal on that corpus. However, ACON achieves this by analyzing errors on a per-example basis, and allowing open-ended prompt optimization from these errors.

While directionally correct, we hypothesize that open-ended prompt optimization based on per-example errors leaves us suboptimally on the high-variance side of the bias/variance tradeoff curve. Instead of open-ended prompt optimization, our crucial insight is that compression errors themselves can be summarized in a low-dimensional set of rules. We then deterministically insert these rules as part of the final summarization prompt, with explicit instructions for our summarization LLM to pay special attention in keeping information from these failure modes in the final compressed text. In other words, rather than freeform prompt optimization guided by noisy single examples, we propose

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

a modular preprocessing step that results in a short descriptive list of common compression errors, which then are used as-is to guide the final compression task.

Our algorithm SENSA (Summarize, Evaluate, Nudge, Summarize Again) starts with a lightweight preprocessing step that summarizes inputs using a vanilla summarization method and produces a list of pertinent information that was lost in the process of that summarization. We then further compress that list of lost information into a small number of information categories, which then is used to update the prompt of the original summarization model. Specifically, the original summarization model is instructed to keep all information that belongs in those categories intact through the summarization process. We show that for fixed token budgets, SENSA results in superior performance in textual summarization compared to ACON and other methods, and does so in a way that is easily human interpreted. For example, SENSA reveals that file and path names are often lost during compression for software engineering datasets. The general schematic for how SENSA operates can be found in Figure 1.

Overall, our contributions within this work are as follows:

- We introduce SENSA (Summarize, Evaluate, Nudge, Summarize Again), a domain-aware algorithm that provides superior summarization performance across a large category of input domains when compared to other state-of-the-art summarization methods like LLMLingua-2 and ACON.
- We show that SENSA not only offers superior performance, but provides human-readable insights into common compression errors within each of our tested domains.
- We show that SENSA's advantage is largest in technical domains like software engineering, where it keeps up to twice as many answerable questions as the next best method.

2 Related Work

Reductive compression. LLMLingua (Jiang et al., 2023) uses a small language model to find the words that are easiest to predict and removes those first, which can shrink text up to twentyfold with little loss in task performance. Selective Context (Li et al., 2023b) applies a similar idea to words, phrases or sentences, and works best when removing whole phrases. LLMLingua-2 (Pan et al., 2024) reframes the problem as token classification, training a small encoder on labels distilled from a larger model; its authors report it as the strongest method in this family that works without seeing the question, and it is the one we compare against. We test both of its released versions, since they give noticeably different results. We leave out LongLLMLingua (Jiang et al., 2024), which chooses words based on the question being asked, because our methods compress before any question is known. Li et al. (2025) survey the wider family.

Error-based compression. A second line of work keeps a language model as the compressor and improves the instruction it is given, building on general prompt-optimization methods (Zhou et al., 2023; Yang et al., 2024; Opsahl-Ong et al., 2024).. ACON (Kang et al., 2026a) and TRACE (Min et al., 2026) both run a loop: compress, look at what went wrong, rewrite the instruction in natural language, and compress again. They differ mainly in what counts as a failure. For ACON it is an agent trajectory that succeeds with the full context but fails with the compressed one; for TRACE it is an action that gets blocked or repeated. SENSA has a similar loop but draws its error signal from directly measuring information retention of the compressed source document. LongSumEval (Nguyen et al., 2026) also uses unanswered questions to guide revisions of a summary.

Evaluation based on information retention. QAGS (Wang et al., 2020) and QuestEval (Scialom et al., 2021) compare answers drawn from a summary and from its source, QAEval (Deutsch et al., 2021) scores a summary by the share of questions from a reference it can answer, and QAFactEval (Fabbri et al., 2022) shows that how the questions are written and filtered matters most. Łajewska et al. (2025) measures how much of the original information survives compression. Kyrkewood (2026) tests gist-based compression against matched questions and finds temporal references especially fragile. Others attach coverage guarantees to retained spans (Kuwahara et al., 2025) or compare formats at one matched budget (Bei et al., 2026), which we expand to multiple budgets in this work. SENSA draws from these evaluation ideas for its primary metric.

Comparisons between reductive vs generative compression. Three recent studies compare the two compression types at several compression ratios. Zakazov et al. (2025) find that off-the-shelf LLMs asked to compress vary widely in how well they follow the requested length, and that a small abstractive compressor trained for the task beats LLMLingua-2 at every ratio they test. SCOPE (Zhang et al., 2025) summarizes a document chunk by chunk and reports its advantage over token-removal methods growing under heavier compression. Jha et al. (2024) find extractive selection generally ahead of both token pruning and summarization. Trained writing compressors such as RECOMP (Xu et al., 2024) and CompAct (Yoon et al., 2024) are trained on question answering data and see the question when they compress, which our setting does not allow. Our comparison adds a rule that every method must actually use the length it was given (§3). A similar budget-dependent choice appears in agent memory: Kang et al. (2026b) show that whether to keep records as they are or to consolidate them depends on the memory budget.

Other summarization methods. Two families of methods fall outside our comparison. Methods that compress context into continuous vectors (Mu et al., 2023; Chevalier et al., 2023; Ge et al., 2024) do not produce text, so a judge cannot read their output and it cannot be charged against a text budget. Methods that train compression jointly with the acting policy (Li et al., 2026) change the model as well as its context, which puts it out of scope of this work as we keep the model static.

3 METHODOLOGY

3.1 SENSA (SUMMARIZE, EVALUATE, NUDGE, SUMMARIZE AGAIN)

Starting with a corpus, compressing it, and measuring the density of questions that could be answered from the original text but are impossible to answer from the information in the compressed text. First, by using the original texts, we use a Qwen3-30B-A3B-Instruct-2507 model to generate questions from the source document, one passage at a time. An LLM judge (Qwen3.5-122B-A10B with no reasoning) then checks each question against its original passage and eliminates any questions that could not be answered from the source document. We check on a small set of data that the judge is well-calibrated and does not vary much with choice of base model (see §4.2). We then compress all source documents with some set compression budget, and the same LLM judge decides whether the questions generated from the full text are still answerable from the compressed document.

A separate model categorizes all questions that are not answerable by the compressed document, and we select the top k ($k = 3$ throughout this manuscript) categories based on the number of failed questions associated with each category. Examples of top k categories can be found in Figure 3. The compression prompt is then augmented with instructions to preserve all text related to the top k categories. See Figure 1 for a schematic of the SENSA algorithm and Algorithm 1 for the formal process description.

3.2 EVALUATION METRIC

The primary metric we measure is the percentage of valid questions that remain answerable by the compressed text output by any method.

The questions generated for each document are split into a development set and a test set. All scores are reported on the test set. Unless otherwise noted, the development set and test set both comprise 50% of the full dataset.

No compressor algorithm is provided access to any of the questions beforehand; compressors are only given visibility to the document and the budget. Judge models are selected specifically to be larger than the compressor models and only see one question at a time. For all prompts related to judge and compressor models, please reference Appendix C. We also provide all LLM models used in Table 1.

3.3 BUDGET AND FILL

All budgets are defined relative to the total length of the original document. To even out passage length distributions, each document goes through the same preprocessing step: each passage is first truncated to 6,000 characters, and questions are generated using that truncated passage. We test SENSA and all other methods at budgets of 4%, 8%, 12%, 16%, and 20%. Budgets of above 20% were deemed to be impractical, as few compression systems are used at such high compression budgets.

Algorithm 1 SENSA. The kinds of content to protect are chosen once, from failures on development documents; compression then needs no questions, labels or judge.

Require: development documents $\mathcal{D}$; document to compress x; budget fraction ρ; tokenizer τ; summarizer S; judge R; inducer I; number of kinds k
Ensure: compressed text y with $\tau(y) \le \rho\,\tau(\mathrm{render}(x))$

Design, run once
for each $d \in \mathcal{D}$ **do**
  $u_d \leftarrow \mathrm{render}(d)$; $s_d \leftarrow S(u_d,\ \rho\,\tau(u_d))$ ▷ plain summary
  $Q_d \leftarrow$ development-half questions written from u_d and answerable from it
end for
$F \leftarrow \{\, q \in Q_d : d \in \mathcal{D},\ R(s_d, q) \text{ is not correct} \,\}$ ▷ questions the plain summary lost
$L \leftarrow$ the k largest kinds in $I(F)$, one rule each ▷ frozen from here on
Compress, one pass per document
$u \leftarrow \mathrm{render}(x)$; $B \leftarrow \rho\,\tau(u)$
$(c_1, \ldots, c_m) \leftarrow$ split u at headings or paragraph breaks, never inside a paragraph
merge neighbouring sections until every $B_i = B\,\tau(c_i)/\tau(u) \ge 150$
for $i = 1, \ldots, m$ **do**
  $y_i \leftarrow S(c_i,\ B_i,\ L)$ ▷ summarize each section on its own share
end for
return $y \leftarrow y_1 \,\|\, \cdots \,\|\, y_m$ ▷ joined in document order

Table 1: Models by role in the answerability measurement. Public model names; the served identifiers and revisions are recorded in each run's artifact.

Role	Model
Summarizer (every writing method)	Qwen3-30B-A3B-Instruct-2507 (Qwen Team, 2025)
Question generator	Qwen3-30B-A3B-Instruct-2507 (Qwen Team, 2025)
Judge (floor check and scoring)	Qwen3.5-122B-A10B (Qwen Team, 2026), reasoning off
Second judge (§4.2)	gpt-oss-120b (OpenAI, 2025)
Category inducer (SENSA's loop)	DeepSeek-V4.1-Flash (DeepSeek-AI, 2026), reasoning off
Guideline optimizer (ACON)	DeepSeek-V4.1-Flash (DeepSeek-AI, 2026), reasoning off
LLMLINGUA-2 (XLM-R)	`xlm-roberta-large` (Conneau et al., 2020)
LLMLINGUA-2 (mBERT)	`bert-base-multilingual` (Devlin et al., 2019)
Token counting	the judge's tokenizer

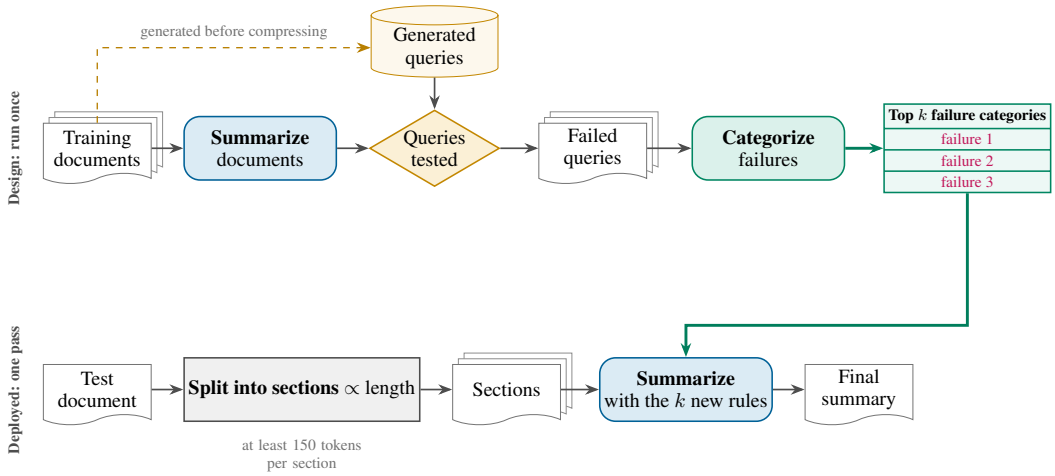
All budgets are treated initially as ceilings, because generative models are not proficient at respecting an exact token budget with precision. Generative methods will usually produce fewer tokens than the allocated budget. We will subsequently ask the compression model to expand the generated text until it is within 10% of the desired budget. See Appendix C.6 for the SENSA prompt that accomplishes this filling procedure. Note that reductive methods (see Section 3.4) are much more precise at respecting our desired compression budget, and thus do not need the additional nudge to expand the output. Appendix B shows how much of its budget each method uses.

3.4 BASELINES

We now detail the SOTA methods that we compare SENSA to for compression performance. We split this section into *generative* methods that produce the compressed text from scratch, and *reductive* methods that directly delete tokens from the original text until it is reduced down to the desired budget.

These two types of compression algorithms are not often compared directly as they tend to have orthogonal evaluation standards; reductive methods measure performance drops as a function of deletion fraction, while generative methods directly measure task performance improvement. In this paper we bring the two method classes into the same evaluation framework by focusing on explicit information retention from the original text.

Reductive methods. LLMLINGUA-2 (XLM-R) (Pan et al., 2024) scores every word of the document and keeps the highest-scoring ones, in their original order, until the budget runs out. It has two released versions. By LLMLINGUA-2 (XLM-R) we always mean the larger one (`xlm-roberta-large`); the smaller one (`bert-base-multilingual`) we call LLMLINGUA-2 (mBERT) and name wherever it appears.



Before: plain summary, whole document

```
Summarize the following government report in at most {BUDGET}
tokens.  Write the most important content first.  Write continuous
prose; do not label content by kind.

---
{DOCUMENT}
---

Summary:
```

After: SENSA, one section

```
Summarize the following government report in at most {BUDGET}
tokens.  Write the most important content first.  Write continuous
prose; do not label content by kind.

In addition, preserve the content of the following kinds, giving
them priority when space is scarce:
- DEFINITION OR SCOPE STATEMENT: A stated meaning, boundary,
inclusion, exclusion, or condition that defines what something is
or is not.
- IDENTIFIER OR SYMBOL NAME: The name, title, label, or designation
assigned to a person, role, organization, system, program,
document, table, account, or entity.
- QUANTITATIVE VALUE OR MEASURE: A numeric amount, rate,
percentage, count, duration, date, threshold, or other measurable
quantity.
When kinds conflict for space, keep the more specific instance.
You MUST stay within the token budget stated above --- preserve by
choosing what to keep inside the budget, never by exceeding it.
---
{DOCUMENT}
---

Summary:
```

Figure 1: *(Top) Schematic flow for SENSA (Summarize, Evaluate, Nudge, Summarize Again):* a vanilla compression algorithm is used on source documents and the resultant summaries are checked by a judge against questions generated from the source before compression; the questions that no longer are answerable by compression are grouped into categories, and the top k are frozen ($k = 3$ in all our experiments). During inference, the test document is split into sections and each section is independently compressed to the desired compression budget with a new SENSA compression prompt that asks to preserve information belonging to the failure categories found previously. The sections are then merged to produce the final result. No questions, labels or judge are necessary at compression time. *(Bottom) Prompt modification example with SENSA.* Example of SENSA prompt modification for a government report corpus. On the left is the plain prompt, while on the right is the modified prompt with the SENSA contribution highlighted.

Generative methods. The writing methods evaluated in this paper all use the same summarizer and differ only in instructions. The plain summary is given a length only. ACON is our reimplementation of ACON (Kang et al., 2026a): an optimizer model reads the questions a plain summary lost on the development half and rewrites a free-text guideline for the summarizer, keeping the best of four candidates. SENSA (Figure 1) splits the document into sections, at headings or paragraph breaks, never inside a paragraph, giving each section a share of the budget in proportion to its length, with at least 150 tokens per section, and summarizes each section on its own with a list of k kinds of content to protect. We use $k = 3$. The kinds are learned from the questions a plain summary lost, as presented in the figure. No method sees a task objective or a label.

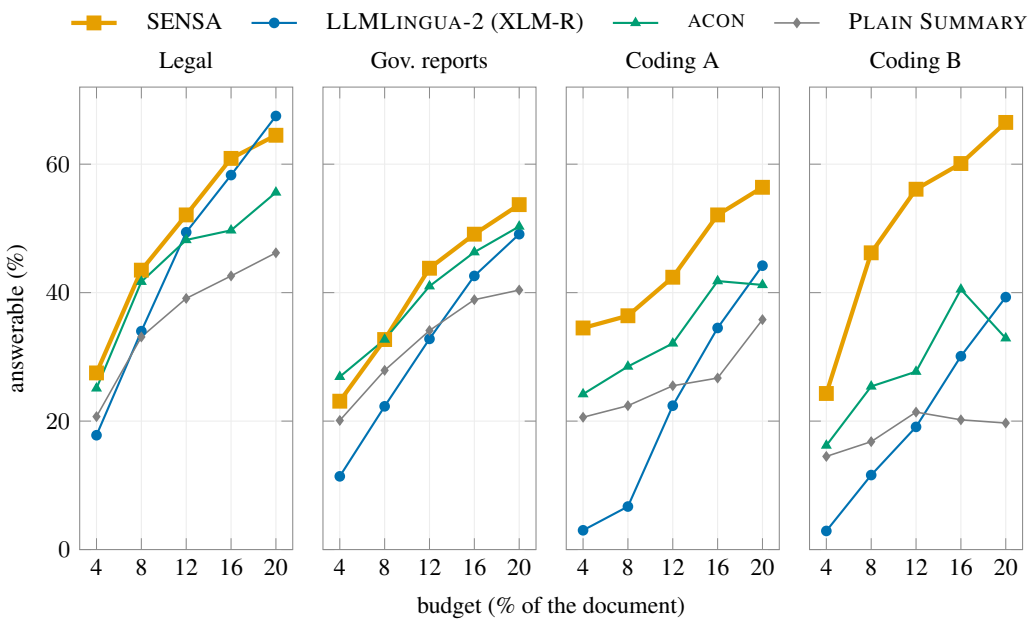


Figure 2: Share of questions still answerable at budgets from 4% to 20% of the document in four domains. SENSA (orange) stays ahead of LLMLINGUA-2 (XLM-R) (blue) at every budget on government reports and both coding datasets. On legal judgments LLMLINGUA-2 (XLM-R) (blue) is ahead at higher budget like 20%.

3.5 DATASETS

We use four document sets that cover a good range of domains. The question counts presented are for the evaluation half. **Legal**: 42 judgments of the Indian Supreme Court from 1950 to 2024 (sinhal, 2026), with 338 questions. **Coding**: two sets of OpenHands (Wang et al., 2025) agent trajectories on SWE-bench tasks (Jimenez et al., 2024), 19 written by Claude 3.5 Sonnet (Coding A, 165 questions) and 21 by Claude 3.5 Haiku (Coding B, 173). **Government reports**: 42 GAO and CRS reports from GovReport (Huang et al., 2021), chosen across lengths, with 324 questions.

4 RESULTS

We compare SENSA against a plain summary, ACON, LLMLINGUA-2 (XLM-R) on four datasets at budgets from 4% to 20% of each source document. Figure 2 and Table 2 provide our primary results. Figure 3 shows examples of error categories that SENSA learns to retain by domain.

4.1 SENSA VS SOTA

Our main results are in Figure 2 and Table 2. SENSA consistently outperforms a plain summary across every domain and every budget. Compared to LLMLINGUA-2 (XLM-R), SENSA does exceptionally well at tight budgets and in technical domains like coding. At extreme compression (4% or 8%) LLMLINGUA-2 (XLM-R) preserves almost none of the answerable content on coding datasets. ACON does relatively well against LLMLINGUA-2 (XLM-R) on verbose domains like legal judgments, and on government reports it comes close to SENSA, edging ahead at 4%. It underperforms in the technical domains. SENSA consistently outperforms these SOTA techniques across all budgets in the technical domain while maintaining its legibility. SENSA single pass is SENSA without the segmentation step, which provides an ablation signal that this segmentation step boosts performance considerably. On legal judgments as budget increases, LLMLINGUA-2 (XLM-R) passes SENSA at 20% but at that level, compression is very mild and LLMLINGUA-2 (XLM-R)'s output is just a string of fragments rather than readable text.

We note that the above results are actually conservative with respect to SENSA. As described in Section 3.4, generative compression methods can lack precision when provided with a specific token budget target. Although we nudge all our generative methods to fill the output window until the outputs come within 10% of

Table 2: Accuracy (%) of questions on compressed set for various compression budgets and domains; the best in each row is in bold. SENSA wins in almost all cases and performs especially well for coding. All results report standard error of less than 1%.

Domain	Budget	LLMLingua-2 Acc. (%) ↑	ACON Acc. (%) ↑	Plain Summary Acc. (%) ↑	SENSA Single Pass Acc. (%) ↑	SENSA Acc. (%) ↑
Legal	4%	17.8	25.1	20.7	26.3	**27.5**
Gov. reports	4%	11.4	**26.9**	20.1	21.9	23.1
Coding A	4%	3.0	24.2	20.6	29.7	**34.5**
Coding B	4%	2.9	16.2	14.5	23.1	**24.3**
Legal	8%	34.0	41.7	33.1	38.8	**43.5**
Gov. reports	8%	22.3	32.7	27.9	**34.8**	32.7
Coding A	8%	6.7	28.5	22.4	**37.6**	36.4
Coding B	8%	11.6	25.4	16.8	30.1	**46.2**
Legal	12%	49.4	48.2	39.1	44.1	**52.1**
Gov. reports	12%	32.8	41.0	34.1	39.3	**43.8**
Coding A	12%	22.4	32.1	25.5	36.4	**42.4**
Coding B	12%	19.1	27.7	21.4	41.6	**56.1**
Legal	16%	58.3	49.7	42.6	51.8	**60.9**
Gov. reports	16%	42.6	46.3	38.9	44.1	**49.1**
Coding A	16%	34.5	41.8	26.7	40.6	**52.1**
Coding B	16%	30.1	40.5	20.2	41.0	**60.1**
Legal	20%	**67.5**	55.6	46.2	50.6	64.5
Gov. reports	20%	49.1	50.3	40.4	47.8	**53.7**
Coding A	20%	44.2	41.2	35.8	43.6	**56.4**
Coding B	20%	39.3	32.9	19.7	38.7	**66.5**

Table 3: Model calls to learn SENSA's error categories (once per domain) and ACON's guidelines (once per budget).

Domain	SENSA	ACON, per budget	ACON, total	SENSA LLM call reduction
Legal	1,116	2,018–2,114	6,185	82%
Gov. reports	1,206	1,976–2,153	6,176	80%
Coding	1,126	1,509–1,750	11,403	90%

the desired budget, we see in Appendix B that on legal judgments SENSA uses slightly less of its budget than the other methods, especially at tighter budgets. Although this is not explicitly reflected in our results within Figure 2 and Table 2, it does mean that our results are likely a slight underestimate of SENSA's true accuracy at various budgets.

SENSA does particularly well at technical domains like coding, where we see large relative improvements against all other methods at every budget point. Relative improvements on coding can be as high as 100% over the next SOTA method, ACON, for a budget of 12% on the second coding dataset (agent trajectories written by Claude-3.5 Haiku on SWE-bench tasks). We hypothesize that summarization errors within technical corpuses like coding are particularly well-described by a small number of error categories, which for coding involve file/path locations, literals/strings, and conditions/control flows (Figure 3).

It is also worth noting that SENSA is quite efficient. Compared to learning ACON's guidelines, learning categories using SENSA takes 80-90% fewer model calls on the same base LLM summarization model. Even for a single budget, SENSA needs about 45% fewer calls than ACON, and it learns its categories only once, whereas ACON must learn a new guideline for every budget (Table 3).

4.2 Judge Variability

To ensure SENSA's scores do not depend on the judge model that was used, we re-scored SENSA summaries at 8% and 16% in every domain with a second judge from a different model family, gpt-oss-120b (OpenAI,

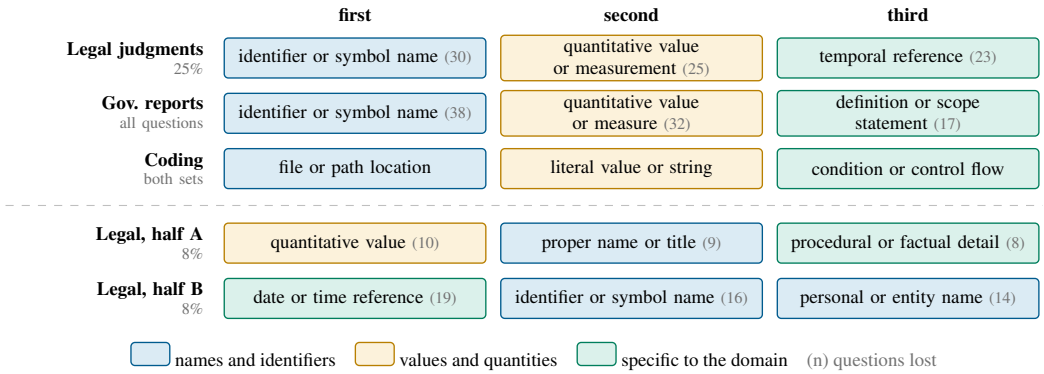


Figure 3: The top $k = 3$ kinds of content SENSA's loop picks in each domain, ranked by how many questions a plain summary lost.

Table 4: Agreement between the two judges on SENSA's summaries. Questions: re-scored by both judges. Chance: agreement expected by chance; κ: agreement beyond chance (1 is perfect).

			Answerable (%)		Agreement (%)		
Domain	Budget	Questions	Judge 1	Judge 2	Observed	Chance	κ
Legal	8%	188	40.4	43.6	96.8	51.2	0.94
	16%	179	59.2	60.3	96.6	51.9	0.93
Gov. reports	8%	195	25.1	27.7	95.4	61.1	0.88
	16%	186	43.0	43.5	95.2	50.9	0.90
Coding A	8%	122	31.1	32.8	95.1	56.5	0.89
	16%	83	53.0	51.8	94.0	50.1	0.88
Coding B	8%	123	39.8	43.1	91.9	51.4	0.83
	16%	114	57.9	57.9	98.2	51.2	0.96

2025). The same prompt was used on a subset of the questions. The two judges agree on 92–98% of verdicts, with Cohen's κ between 0.83 and 0.96 (Table 4). The second judge is usually slightly more lenient, but its score for SENSA is never more than 3.3 points away from the first judge's.

5 Discussion

SENSA shows superior performance, but beyond its raw performance SENSA also provides a number of important benefits for textual compression that are relatively noteworthy:

- **Human Interpretability:** Because SENSA begins by condensing missing information categories from a domain-specific corpus, we first obtain human-readable insights into the difficult parts of any particular domain (e.g. file and directory names for software-related inputs). This interpretability provides value not just because it provides confidence that the intermediate state of the algorithm is appropriate, but because it provides an additional point of entry where we can potentially inject additional information into the summarization prompt for improved control (see Future Work below). In addition, in contrast to reductive compression methods like LLMLingua-2 (XLM-R) (Pan et al., 2024), the summaries output by SENSA also retain the same level of human readability as the source documents.
- **High performance on technical domains:** While SENSA shows state-of-the-art performance across many domains, it enjoys particularly robust performance on technical domains like coding. A possible reason is that technical documents have relatively spiky distributions for compression errors. A small number of error categories explains a large percentage of the variance within these domains.

Limitations and Future Work We have shown that SENSA works quite well at restrictive compression budgets. However, at higher budgets SENSA performance does lag behind some compression algorithms that operate through token deletion versus token generation. We find that the performance in the high-budget limit is challenging for generative compression algorithms as generative methods have trouble adhering to token limits

SENSA

opening
The F-35 Lightning II program, the Department of Defense's most costly weapon system [. . .]

answers to questions 3 and 6
ALIS 3.5, a "stabilization" release, still contained eight category 1 deficiencies [. . .] **a backlog of 4,700 open deficiencies as of September 2019** [. . .]

last words
[. . .] DOD's proposed follow-on modernization strategy is incremental but not yet finalized. [. . .]

7/8 answered
2,136 of 2,249 tokens, 9 sections

SENSA Single Pass

opening
The F-35 Lightning II aircraft, a joint multinational acquisition program [. . .]

answers to questions 3 and 6
(no sentence containing either answer)

last words
[. . .] Despite the existence of the Action Request (AR) process, there is no centralized, real-time dashboard or tracking system that provides visibility into the status of all open ARs across the F-35 enterprise.

2/8 answered
2,244 of 2,249 tokens

ACON

opening
The F-35 Lightning II program, a joint multinational acquisition initiative [. . .]

answers to questions 3 and 6
(no sentence containing either answer)

last words
[. . .] The future of the F-35 depends on resolving ALIS's systemic challenges through coordinated leadership, transparency, and a commitment to modern, agile, and user-driven software development.

3/8 answered
2,142 of 2,249 tokens

LLMLINGUA-2 (XLM-R)

opening
F-35 Lightning Force Marine Corps costly $1.6 trillion 66 Logistics System [. . .]

the same facts, as fragments
[. . .] heavy servers scheduling delays deficiencies updates 18 **4,700 open ALIS deficiencies** deficiencies testing4,700 34 percent 2017 22 percent 1 2 1 [. . .]

last words
[. . .] GAO-15-364 Affordable Strategy Cost Estimates GAO-14-778 Software Restructuring Affordability Challenges Risks

3/8 answered
2,049 of 2,249 tokens

All eight questions for this report (shortened)	SENSA	One-pass	ACON	LLMLINGUA-2 (XLM-R)
1 Which category of deficiencies is critical?	✓	–	–	–
2 When did officials say metrics were still being developed?	–	–	–	–
3 What is the "stabilization" release called?	✓	–	–	–
4 Which system is not performing well?	✓	✓	✓	✓
5 Which system is hard to deploy to forward locations?	✓	✓	✓	✓
6 How many deficiencies were open in September 2019?	✓	–	–	–
7 What servers are needed to collect aircraft data?	✓	–	✓	–
8 What support is critical for deployments at two sites?	✓	–	–	✓

Figure 4: A government report (GAO-20-316) compressed to 12% of its length (2,249 tokens), text verbatim with omissions marked. The three writers open almost identically but end in different places: SENSA, which gives each of its nine sections its own share of the budget, ends on the report's closing pages; one-pass SENSA, with the same instruction applied to the whole report, runs out in the middle of the findings; ACON ends on closing remarks of its own. Only SENSA states the answers to questions 3 and 6 (highlighted). LLMLINGUA-2 (XLM-R) keeps the number 4,700 too, but inside a run of fragments with no sentence around it, and the judge cannot use it. The table lists every question for this report. We chose the report as the one with SENSA's largest margin over one-pass SENSA among the 42 (5 questions won, none lost; the next best are +3, +3 and +2), so it is the best case, not a typical one: across all 42 reports at this budget the gain is $+3.1$ over one-pass SENSA and $+6.5$ over LLMLINGUA-2 (XLM-R), neither clear of zero.

when the token limits are large. Possible future work may focus on better control over budget limits, which may also improve performance of generative compression algorithms like SENSA within these higher token regimes. Another weakness of SENSA is that it is not sensitive to downstream task success. Better control may also allow SENSA to differentially focus on specific categories that improve success rate at whatever the end goal is of the overall AI system.

6 CONCLUSION

Within this work, we introduced SENSA (Summarize, Evaluate, Nudge, Summarize Again), a novel and intuitive compression algorithm that achieves state-of-the-art performance for textual summarization within both technical and non-technical domains. SENSA is fundamentally anchored to the domain of the textual input, and involves a preprocessing step where common summarization errors within that domain are analyzed and condensed into a low-dimensional set of rules. SENSA's internal logic is human-interpretable, and it is especially performant within technical domains.

As modern language models continue to advance, accurate and efficient compression algorithms are becoming more and more necessary to deal with exploding context windows and increasing token costs. Given that language models are also becoming increasingly specialized for certain tasks and domains, compression algorithms should also be able to reason about the properties specific to those domains. To that end, SENSA presents a sensible compression technique that condenses the relevant information of a whole domain to use as its primary guardrail, which leads to impressive performance across the board.

Reproducibility Statement

The measurement instrument is specified in §3.1 and the methods compared in §3.4. Appendix A gives the pinned model identifiers by role, the compressor package version and checkpoint revision, the prompt hashes recorded per run, and the guards that cause a run to refuse to emit a number. Appendix C prints every prompt word for word.

Use of Large Language Models

Language models are the object of study in this paper: they are the compressors, the question generators, the readers and the judges, and §3.1 and Appendix A specify which model fills which role.

Language models were used as a coding and analysis assistant: writing and reviewing the experiment runners, the statistical analysis code and the figure code, and reviewing registrations and results for defects. Every number reported in the paper is produced by tracked code from recorded artifacts, and every claim was checked against those artifacts by the authors, who take full responsibility for the content. Language models were used to help surface related work the authors may have missed, but all references were verified by the authors against the original sources. Language models were also used for grammar checks and occasional rewording help in preparation of the manuscript, but made no decisions about the claims made in the work.

References

Sisong Bei, Mikhail L. Arbuzov, Ziwei Dong, Dmitri Kalaev, and Alexey Shvets. Context compression is not one thing: Readable symbolic re-expression vs. coherent summary at matched budget, 2026. URL https://arxiv.org/abs/2606.14875.

Alexis Chevalier, Alexander Wettig, Anirudh Ajith, and Danqi Chen. Adapting language models to compress contexts. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 3829–3846, 2023.

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. Unsupervised cross-lingual representation learning at scale. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 8440–8451, 2020.

DeepSeek-AI. DeepSeek-V4.1-Flash: Pushing the limits of KV cache compression. Technical report, 2026. URL https://huggingface.co/deepseek-ai/DeepSeek-V4.1-Flash.

Daniel Deutsch, Tania Bedrax-Weiss, and Dan Roth. Towards question-answering as an automatic metric for evaluating the content quality of a summary. *Transactions of the Association for Computational Linguistics*, 9:774–789, 2021.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 4171–4186, 2019.

Alexander Fabbri, Chien-Sheng Wu, Wenhao Liu, and Caiming Xiong. QAFactEval: Improved QA-based factual consistency evaluation for summarization. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL)*, pp. 2587–2601, 2022.

Tao Ge, Jing Hu, Lei Wang, Xun Wang, Si-Qing Chen, and Furu Wei. In-context autoencoder for context compression in a large language model. In *International Conference on Learning Representations (ICLR)*, 2024.

Luyang Huang, Shuyang Cao, Nikolaus Parulian, Heng Ji, and Lu Wang. Efficient attentions for long document summarization. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 1419–1436, 2021. doi: 10.18653/v1/2021.naacl-main.112.

Siddharth Jha, Lutfi Eren Erdogan, Sehoon Kim, Kurt Keutzer, and Amir Gholami. Characterizing prompt compression methods for long context inference. In *ICML 2024 Workshop on Efficient Systems for Foundation Models (ES-FoMo II)*, 2024. arXiv:2407.08892.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. LLMLingua: Compressing prompts for accelerated inference of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 13358–13376, 2023.

Huiqiang Jiang, Qianhui Wu, Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. LongLLMLingua: Accelerating and enhancing LLMs in long context scenarios via prompt compression. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1658–1677, 2024.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations*, 2024.

Greg Kamradt. Needle in a haystack: Pressure testing LLMs. GitHub repository, 2023. URL https://github.com/gkamradt/LLMTest_NeedleInAHaystack.

Minki Kang, Wei-Ning Chen, Dongge Han, Huseyin A. Inan, Lukas Wutschitz, Yanzhi Chen, Robert Sim, and Saravan Rajmohan. ACON: Optimizing context compression for long-horizon LLM agents. In *Proceedings of the 43rd International Conference on Machine Learning (ICML)*, 2026a. arXiv:2510.00615.

Qingcan Kang, Mingyang Liu, Shixiong Kai, Kaichao Liang, Zhentao Tang, Yuqi Cui, Tao Zhong, and Mingxuan Yuan. Retain or consolidate? budget-dependent operator selection for language agent memory, 2026b. URL https://arxiv.org/abs/2607.17545.

Bruce Kuwahara, Chen-Yuan Lin, Xiao Shi Huang, Kin Kwan Leung, Jullian Arta Yapeter, Ilya Stanevich, Felipe Perez, and Jesse C. Cresswell. Document summarization with conformal importance guarantees. In *Advances in Neural Information Processing Systems*, 2025.

Nicholas E. Kyrkewood. The sleeping agent: What gist-based context compression loses and why, 2026. URL https://arxiv.org/abs/2608.11775.

Weronika Łajewska, Momchil Hardalov, Laura Aina, Neha Anna John, Hang Su, and Lluís Màrquez. Understanding and improving information preservation in prompt compression for LLMs. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 17520–17541, Suzhou, China, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.findings-emnlp.949. URL https://aclanthology.org/2025.findings-emnlp.949/.

Yinheng Li, Shaofei Wang, Han Ding, and Hang Chen. Large language models in finance: A survey. In *Proceedings of the Fourth ACM International Conference on AI in Finance (ICAIF)*, pp. 374–382, 2023a.

Yucheng Li, Bo Dong, Frank Guerin, and Chenghua Lin. Compressing context to enhance inference efficiency of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 6342–6353, 2023b. doi: 10.18653/v1/2023.emnlp-main.391.

Yujiang Li, Zhenyu Hou, Yi Jing, Jie Tang, and Yuxiao Dong. CompactionRL: Reinforcement learning with context compaction for long-horizon agents, 2026. URL https://arxiv.org/abs/2607.05378.

Zongqian Li, Yinhong Liu, Yixuan Su, and Nigel Collier. Prompt compression for large language models: A survey. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 7182–7195, 2025.

Guanghui Min, Liang Wu, Mayank Darbari, Chen Chen, and Liangjie Hong. Toward reliable context compression for long-horizon agents: An empirical study of execution instability, 2026. URL https://arxiv.org/abs/2608.06503.

Ali Modarressi, Hanieh Deilamsalehy, Franck Dernoncourt, Trung Bui, Ryan A. Rossi, Seunghyun Yoon, and Hinrich Schütze. NoLiMa: Long-context evaluation beyond literal matching. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, volume 267 of *Proceedings of Machine Learning Research*, pp. 44554–44570, 2025.

Jesse Mu, Xiang Lisa Li, and Noah Goodman. Learning to compress prompts with gist tokens. In *Advances in Neural Information Processing Systems*, 2023. URL https://arxiv.org/abs/2304.08467.

Huyen Nguyen, Haoxuan Zhang, Yang Zhang, Junhua Ding, and Haihua Chen. LongSumEval: Question-answering based evaluation and feedback-driven refinement for long document summarization, 2026. URL https://arxiv.org/abs/2604.25130. IEEE AITest 2026.

OpenAI. gpt-oss-120b & gpt-oss-20b model card, 2025. URL https://arxiv.org/abs/2508.10925. Model: https://huggingface.co/openai/gpt-oss-120b.

Krista Opsahl-Ong, Michael J. Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. Optimizing instructions and demonstrations for multi-stage language model programs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 9340–9366, 2024.

Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Menglin Xia, Xufang Luo, Jue Zhang, Qingwei Lin, Victor Rühle, Yuqing Yang, Chin-Yew Lin, H. Vicky Zhao, Lili Qiu, and Dongmei Zhang. LLMLingua-2: Data distillation for efficient and faithful task-agnostic prompt compression. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 963–981, 2024. doi: 10.18653/v1/2024.findings-acl.57.

Qwen Team. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388. Model: https://huggingface.co/Qwen/Qwen3-30B-A3B-Instruct-2507.

Qwen Team. Qwen3.5: Towards native multimodal agents. Blog post, February 2026. URL https://qwen.ai/blog?id=qwen3.5. Model: https://huggingface.co/Qwen/Qwen3.5-122B-A10B.

Thomas Scialom, Paul-Alexis Dray, Sylvain Lamprier, Benjamin Piwowarski, Jacopo Staiano, Alex Wang, and Patrick Gallinari. QuestEval: Summarization asks for fact-based evaluation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 6594–6604, 2021.

Karan Singhal, Shekoofeh Azizi, Tao Tu, S. Sara Mahdavi, Jason Wei, Hyung Won Chung, et al. Large language models encode clinical knowledge. *Nature*, 620:172–180, 2023.

sinhal. Indian Supreme Court Judgments. Hugging Face dataset, 2026. URL https://huggingface.co/datasets/sinhal/Indian_Supreme_Court_Judgments. Licence: OpenRAIL; source: https://main.sci.gov.in.

Dave Van Veen, Cara Van Uden, Louis Blankemeier, Jean-Benoit Delbrouck, Asad Aali, Christian Bluethgen, Anuj Pareek, Malgorzata Polacin, Eduardo Pontes Reis, Anna Seehofnerová, Nidhi Rohatgi, Poonam Hosamani, William Collins, Neera Ahuja, Curtis P. Langlotz, Jason Hom, Sergios Gatidis, John Pauly, and Akshay S. Chaudhari. Adapted large language models can outperform medical experts in clinical text summarization. *Nature Medicine*, 30:1134–1142, 2024.

Alex Wang, Kyunghyun Cho, and Mike Lewis. Asking and answering questions to evaluate the factual consistency of summaries. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 5008–5020, 2020.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. OpenHands: An open platform for AI software developers as generalist agents. In *International Conference on Learning Representations*, 2025.

Shijie Wu, Ozan Irsoy, Steven Lu, Vadim Dabravolski, Mark Dredze, Sebastian Gehrmann, Prabhanjan Kambadur, David Rosenberg, and Gideon Mann. BloombergGPT: A large language model for finance, 2023. URL https://arxiv.org/abs/2303.17564.

Fangyuan Xu, Weijia Shi, and Eunsol Choi. RECOMP: Improving retrieval-augmented LMs with context compression and selective augmentation. In *The Twelfth International Conference on Learning Representations*, 2024.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *International Conference on Learning Representations (ICLR)*, 2024.

Chanwoong Yoon, Taewhoo Lee, Hyeon Hwang, Minbyul Jeong, and Jaewoo Kang. CompAct: Compressing retrieved documents actively for question answering. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 21424–21439, 2024.

Ivan Zakazov, Alexander Sharipov, Berke Argın, Oussama Gabouj, Kamel Charaf, Alexi Semiz, Lorenzo Drudi, Nicolas Baldwin, and Robert West. Cmprsr: Abstractive token-level question-agnostic prompt compressor, 2025. URL https://arxiv.org/abs/2511.12281.

Tinghui Zhang, Yifan Wang, and Daisy Zhe Wang. SCOPE: A generative approach for LLM prompt compression, 2025. URL https://arxiv.org/abs/2508.15813.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. In *International Conference on Learning Representations (ICLR)*, 2023.

A REPRODUCIBILITY

Models by role. Every model is pinned by its exact served identifier, with reasoning switched off, and each was tested on a full-length prompt before we used it. Table 1 in §3.1 lists the model used in each role.

Who writes the questions. The questions are written by the same model that writes every summary. This could give the writing methods an edge over LLMLINGUA-2 (XLM-R) and LLMLINGUA-2 (mBERT), which use no language model: if the model's sense of what is worth asking matches its sense of what is worth keeping, its own summaries will tend to keep the answers. Comparisons among the writing methods (SENSA, ACON and the plain summary) all share this and are unaffected by it. Comparisons between writing and deleting, including where the two cross, are not, and should be read with this in mind. A check with questions written by a different model family would settle it; we have not yet run one.

The judge and the summarizer share a family. Both are Qwen models, so the judge might favour text written the way it would write. Section 4.2 re-scores SENSA's outputs with a judge from a different model family.

The classifier. We use `llmlingua==0.2.2`, with the checkpoint named by repository and revision hash, no access to the questions, the target length re-counted in the judge's tokenizer with at most four adjustment passes, and no forced tokens. Our code checks which checkpoint actually loaded before it processes the first document.

Prompts. Each run records the SHA-256 hash of every prompt it used, so we can verify that the prompt that ran is the one printed in Appendix C. Prompts are kept in files rather than written into the code, for the same reason.

Call logs and resuming. Every model call is written to a per-run log with its role, prompt name and version. Generated texts and verdicts are saved as they are produced, so an interrupted run picks up where it stopped without paying for the same calls twice. A separate file marks a run as finished, so a run that stopped at a guard cannot be mistaken for one that completed.

Guards that stop a run. A run stops instead of reporting a number if fill falls outside the fill band, if the share of stopwords rises too far above the method's own baseline, if the wrong classifier checkpoint loads, if a question cites a passage that does not exist, if a question is identical to the generator's template, or if a verdict cannot be parsed.

Document sets. Each run records its document identifiers, the random seed used to draw them, and every exclusion with its reason.

B BUDGET UTILIZATION

Every output we score must use between 90% and 100% of its budget (§3), so that methods are compared at nearly the same length. Figure 5 shows how each method meets this rule on legal judgments, the one domain where every method we report was run at every budget.

SENSA stays in the band in every domain and at every budget. Its lowest average, on legal judgments at 4%, sits on the lower edge at 0.90; everywhere else it is higher, and on the coding sets it is close to 1.0. Because each section gets a small budget of its own, the summarizer fills each one, and SENSA's output keeps pace with the budget all the way to 20%.

A summary written in one pass behaves differently. On the prose domains its length keeps up with the budget until about 11–13% of the document and then grows more slowly than the budget does, because the summarizer runs out of new things to say before it reaches the limit. On the coding transcripts it keeps up all the way to 20%. This is why the plain summary and ACON use a little less of their budget at 16% and 20% than at lower budgets, although on legal judgments they stay inside the band.

C PROMPTS

Every prompt that produced a reported number is printed below, word for word, with its placeholders in blue. Three models are involved. The summarizer writes the questions, every summary and every top-up; the larger judge checks the questions against the source and scores every compressed text; a third model learns SENSA's kinds and runs ACON's analyzer and optimizer (Table 1). LLMLINGUA-2 (XLM-R) uses no prompt. Table 5 shows where each prompt is used, in the order a document meets them.

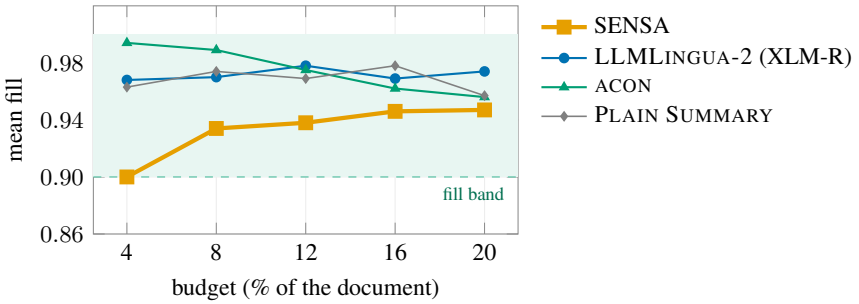


Figure 5: How much of its budget each method used on legal judgments (42 documents), averaged over documents; the shaded strip is the fill band. All four methods stay inside the band at every budget, so every legal comparison in the paper is between outputs of nearly the same length.

Table 5: Where each prompt is used, in the order a document meets them.

Step	Model	Prompt
Write questions from each passage	summarizer	§C.1
Check each question against the source	judge	§C.2
Learn the kinds (once per domain)	inducer	§C.3
Compress: plain summary, SENSA	summarizer	§C.4
Learn ACON's guideline (once per domain)	inducer	§C.5
Compress: ACON	summarizer	§C.5
Top up a summary under 90% of its budget	summarizer	§C.6
Score the compressed text	judge	§C.2

C.1 WRITING THE QUESTIONS

The summarizer writes four questions per passage ({k} is 4; {item_text} is the passage after truncation). The prompt was written for coding sessions and was used unchanged in every domain, which is why it speaks of a software agent and a tool observation.

Question writing

```
You are constructing probe questions to test whether a compacted session summary preserved a
specific piece of information a software-engineering agent observed earlier.

Below is ONE tool observation from that session.

Write exactly {k} questions.  EVERY question must satisfy ALL of these:

CONTENT
1.  Answerable using ONLY the observation below.  No outside knowledge.
2.  About what the agent LEARNED --- a value, a returned path, an error message, a symbol name, a
   state of the code or system.
3.  NOT about how the tool was invoked.  Never ask which function or command was used, what its
   arguments were, or how the call was written.  A summary should drop that, and losing it costs
   the agent nothing.

VERIFIABILITY --- the answer must be checkable by someone holding only this observation:
4.  The answer must be a LITERAL STRING that appears in the observation --- a name, a path, a
value,
   a message.  Not something inferred, summarised, or computed.
5.  NO counting, totals, tallies or aggregates ("how many...", "what is the total...").  These are
   not verifiable at a glance and are the main source of unanswerable probes.
6.  NO questions that require scanning or comparing the whole observation ("which is the largest",
   "what comes last", "are there any...").  Ask about ONE named thing.
7.  If the observation is a long listing (a directory tree, a file index, search results), pick a
   SPECIFIC named entry and ask something concrete about it --- never ask about the listing as a
   whole.

FORM
8.  Something the agent could plausibly need later in the task, not incidental trivia.
9.  Each targets a DIFFERENT fact.  Do not reveal the answer in the question.

Output exactly {k} lines.  One question per line.  No numbering, no preamble, no blank lines.

OBSERVATION:
---
{item_text}
---
```

C.2 CHECKING AND SCORING

The judge uses one prompt for both jobs. To check a question, the whole source goes in the summary slot, and the question is kept only if the verdict is ANSWERABLE. To score a compressed text, that text goes in the summary slot. The first word of the reply is the verdict; a reply that cannot be read is retried once and otherwise counted as lost, for every method alike.

Checking the source and scoring a compressed text

```
You are checking whether a compacted session summary still contains the information needed to
answer a question.

Begin your response with exactly one word --- ANSWERABLE or LOST --- before anything else.  You may
follow it with at most one short justification sentence.

ANSWERABLE --- the summary contains the specific information the question asks for.
LOST --- the summary does not contain it, or contains only a vague gesture at it.

Judge on MEANING, not wording.  A paraphrase that carries the same specific fact counts as
ANSWERABLE. A summary that mentions the general topic but omits the specific value is LOST.

Do not use outside knowledge.  Do not guess from plausibility.  If the specific fact is not
recoverable from the summary text, answer LOST.

Do not answer the question itself.  Render only the verdict.

SUMMARY:
---
{summary}
---

QUESTION: {question}

Verdict first, one word:
```

C.3 LEARNING THE KINDS

Once per domain, the inducer reads the development-half questions a plain summary lost and proposes four to six kinds of content ({k_min} is 4, {k_max} is 6). A second call assigns each lost question to one kind. The kinds are ranked by how many lost questions they hold, kinds holding under 5% of them are dropped, and the top three become SENSA's list (Figure 3 and Table 6).

Proposing kinds

```
You are analysing which kinds of information a summarisation step DESTROYS.

Below are questions that a compacted summary could no longer answer.  Each one names a fact that
was
present in the original context and did not survive compaction.

Induce between {k_min} and {k_max} CATEGORIES that describe the KINDS of fact being lost.

Hard requirements:
1.  Each category must be a general RULE, applicable to questions you have never seen.  Write
   "identifier or symbol name", not "the RST class name".  Write "error or failure message", not
   "the ImportError on line 42".
2.  NEVER include a filename, file path, line number, specific error class, library name, or any
   number in a category name or definition.  A category containing one is invalid.
3.  Categories must be mutually distinguishable by reading a question alone.
4.  Together they should cover the space of questions, not just the common cases.

Output exactly one category per line:  a short upper-case name, a colon, then the definition.
The two lines below illustrate the LINE SHAPE ONLY. The words they contain are not candidates.

EXAMPLE_ONE: Shows that a line is a name, a colon, then a one-sentence rule.
EXAMPLE_TWO: Shows that several such lines are expected.

Do not output the words "NAME" or "CATEGORY", never start a name with "EXAMPLE_", and never copy an
illustration line.  No numbering, no preamble, no blank lines, no markdown.

QUESTIONS:
{questions}
```

Assigning a lost question to a kind

```
Assign the question below to exactly one category.

CATEGORIES:
{categories}

Answer with the category NAME alone, copied exactly.  If no category fits, answer exactly:
{unassignable}

QUESTION: {question}
```

Table 6: The three kinds SENSA learned for each domain, with the one-line rule each carries into the prompt. They fill the {KINDS} slot in §C.4, one line per kind, in alphabetical order.

Domain	Kind	Rule
Legal judgments	identifier or symbol name	Requests the name, label, or symbol used to designate a specific entity, provision, or item.
	quantitative value or measurement	Requests a numeric amount, dimension, percentage, duration, or other measurable quantity.
	temporal reference	Requests a date, time, or chronological marker associated with an event or document.
Government reports	definition or scope statement	A stated meaning, boundary, inclusion, exclusion, or condition that defines what something is or is not.
	identifier or symbol name	The name, title, label, or designation assigned to a person, role, organization, system, program, document, table, account, or entity.
	quantitative value or measure	A numeric amount, rate, percentage, count, duration, date, threshold, or other measurable quantity.
Coding (both sets)	condition or control flow	A condition, trigger, parameter, or control-flow fact such as a flag value, branch condition, or argument passed.
	file or path location	The name, path, or directory location of a file, module, or resource.
	literal value or string	The exact value of a constant, setting, parameter, or literal string such as a message, key, or assigned value.

C.4 WRITING SUMMARIES

Every summary starts from the same opening line, with {domain} filled in as *legal judgment document*, *government report* or *coding agent transcript*, and {budget} as the token allowance for that call. The budget is also a hard cap on generation, the same for every method. The plain summary sends the whole document once. SENSA sends each section on its own share of the budget and adds the highlighted block, with {KINDS} replaced by the domain's three kinds (Table 6), each written as "– NAME: rule".

Plain summary, whole document

```
Summarize the following {domain} in at most {budget} tokens.  Write the most important content
first.  Write continuous prose; do not label content by kind.
---
{DOCUMENT}
---
Summary:
```

SENSA, one section

```
Summarize the following {domain} in at most {budget} tokens.  Write the most important content
first.  Write continuous prose; do not label content by kind.
In addition, preserve the content of the following kinds, giving them priority when space is
scarce:
{KINDS}
When kinds conflict for space, keep the more specific instance.  You MUST stay within the token
budget stated above --- preserve by choosing what to keep inside the budget, never by exceeding
it.
---
{SECTION}
---
Summary:
```

C.5 ACON

Our reimplementation of ACON learns one guideline per domain. The analyzer reads, for one development document, the plain summary and the questions it did and did not answer, and suggests edits. The optimizer then rewrites the summarizer's prompt from those suggestions; it proposes four candidates, and the one that keeps

the most development-half questions is used, like any other prompt, to write the summaries. The optimizer's instruction to shorten the summary comes from ACON's own optimizer, which was built to cut context length in agent tasks. Here the budget is fixed, and a summary that comes out short is rewritten to length (§C.6).

ACON analyzer, one document

```
You are an expert context compression diagnostician.
Scope: You will receive, separately, (1) the document compressor's input/output for a single pass
and (2) ONLY the document's probe questions for grounding. No other raw logs are included.
Goal: Propose precise, actionable edits that safely reduce the length of "OUTPUT_SUMMARY" of
<<<OPTIMIZED_HISTORY>>> based on the given optimizer input (INPUT_PROMPT) and PROBE_QUESTIONS
info.
Focus on CONTENT in the OUTPUT_SUMMARY, not the formatting. Identify which information can be
dropped or compressed considering its impact on answering the document's probe questions.

Inputs:
- task_name: {task_name}
- orig_chars (chars of optimizer output only): {orig_chars}

Document Compressor IO (single pass):
<<<OPTIMIZED_HISTORY>>>
{optimized_history}
<<<END_OPTIMIZED_HISTORY>>>

Probe Questions (each tagged with whether the summary still answers it):
<<<NEXT_SESSION>>>
{next_session}
<<<END_NEXT_SESSION>>>

Output a focused analysis JSON with concrete recommendations. Base every suggestion only on the
content above; do not invent missing details or request additional context. Wrap the JSON between
the markers below.

Schema:
{
  "overview": {"orig_chars": int, "main_themes": [str]},
  "analysis": str,
  "keep": [{"reason": str, "evidence_spans": ["short quote"]}],
  "remove_or_shorten": [{"category": "repetition|chatter|tool-logs|formatting|meta",
                         "pattern": str, "example_span": str, "action": "drop|summarize",
                         "risk_if_removed": "low|medium|high"}],
  "summarize_rules": [str],
  "critical_refs": [str]
}

<<<ANALYSIS_JSON>>>
{...}
<<<END>>>
```

ACON optimizer, one candidate

```
You are an expert prompt engineer tasked with refining a DOCUMENT SUMMARIZATION prompt.
Rewrite the ORIGINAL PROMPT to reduce length of the DOCUMENT SUMMARY while preserving answerability
of the document's facts under the fixed token budget.
Ground all changes in the PER-SAMPLE REDUCTION SIGNALS below. Do not aggregate across samples; use
the patterns and rules as-is.

Constraints:
- Keep all placeholders, variable names, and structure intact where possible.
- Add explicit, concrete rules that prevent verbosity and retain essential state.
- Do not include literal values from prior content; refer to variable names only.
- Output ONLY the improved prompt template (no extra commentary).

Context (samples below are the only ground truth signals to use):
- Average original summary size (chars) across sampled set: {avg_orig_chars}

{samples}

Original Prompt Template (verbatim between markers):
<<<ORIGINAL_PROMPT>>>
{original_prompt}
<<<ORIGINAL_PROMPT>>>

Output only the improved prompt template text, ready to be used as a summarization instruction.
```

C.6 TOPPING UP SHORT SUMMARIES

A summary that uses less than 90% of its budget gets a top-up. The plain summary gets up to three rounds and SENSA one round per section; each shows the source and the text so far, asks only for detail not already there, and keeps the longest draft that fits the budget. (In every reported run {block_or_nothing} is empty.) ACON instead gets one rewrite, which replaces its draft.

Top-up, plain summary

```
Below is a source and a summary of it.  Write ADDITIONAL detail drawn from the source that is NOT
already present in the summary.  Output only the additional text --- no preamble, no headings, no
repetition of what is already there.
{block_or_nothing}
---
{DOCUMENT}
---
Summary so far:
{SUMMARY}
Additional detail:
```

Top-up, one SENSA section

```
Below is a source and a summary of it.  Write ADDITIONAL detail drawn from the source that is NOT
already present in the summary.  Output only the additional text --- no preamble, no headings, no
repetition of what is already there.
---
{SECTION}
---
Summary so far:
{SUMMARY}
Additional detail:
```

Rewrite, ACON

```
Your summary used only {used_tokens} of the {budget} allowed tokens.  Rewrite it to use close to the
full allowance --- about {budget} tokens --- adding preserved specifics from the source.  Stay at
or under {budget} tokens.
---
{DOCUMENT}
---
Summary:
```