

Thread-Preserving Chunking for LLM-Based Action Item Extraction from Enterprise Email under Constrained Context Budgets

Kiyeop Kim Donghoon Cho

Manufacturing-AI Transformation, LIG Defense & Aerospace (LIG D&A)

Gumi, Republic of Korea

kiyeop.kim@ligdna.com cho.dong.hoon@ligdna.com

Abstract

Enterprises increasingly run small, on-premises large language models (LLMs) to extract action items from email where data-sovereignty rules forbid cloud APIs. On shared GPUs serving many users, the per-request context budget is small, forcing inboxes to be *chunked*. We show that the chunking strategy substantially affects extraction quality. We propose **Thread-Preserving (TP) chunking**, which groups emails by thread and never splits one across chunks, and compare it with **Fixed-Length (FL) chunking** on a synthetic set of 80 Korean enterprise-email cases (385 emails, 93 gold tasks), three models (Qwen2.5-7B, LLaMA-3.1-8B, GPT-OSS-120B), six chunk sizes (500–5,000 chars), and three runs each. At $\leq 1,000$ chars, TP improves F1 by up to **16.2** points and cuts false positives by **40–51%**; gains for the 7B/8B models are about double those of the 120B model. The gap closes as chunks grow. TP is a zero-cost preprocessing step requiring no model change, and is most valuable where hardware is tightest.

1 Introduction

Large language models (LLMs) are widely used to automate routine office work, including extracting action items and meeting schedules from email (Bommasani et al., 2021). Because enterprise email carries sensitive data, many organizations—particularly in defense, finance, and public administration—deploy LLMs *on-premises* inside closed networks rather than via cloud APIs.

On-premises deployment imposes hard constraints. In a multi-user environment sharing limited GPU memory, the effective context window per request is small: although models advertise 32k–128k-token windows, system prompts, few-shot examples, and reserved output tokens can leave only a small fraction for the actual email content. Motivated by this, we study a range of constrained

budgets (500–5,000 characters per chunk). Processing a long inbox under such a budget requires splitting the input into segments—*chunking*.

The dominant approach, fixed-length (FL) chunking, splits text at a fixed character or token count. But email is not prose: a single task may evolve across replies and forwards—a deadline revised, a task canceled—while unrelated threads are interleaved within one inbox. FL chunking is blind to this structure: by cutting threads at arbitrary points, it separates a request from its later modification, which produces a characteristic set of errors: extracting tasks that were subsequently canceled, missing updated deadlines, and emitting the same task from multiple fragments—that is, false positives (hallucinations).

We propose **Thread-Preserving (TP) chunking**, which groups emails into conversation threads (via metadata) and guarantees that no thread is split across chunks, even at the cost of slightly underfilling a chunk. TP can be viewed as a domain-specific instance of structure-aware chunking, instantiated for the metadata structure of email rather than continuous prose. Our central contribution is accordingly the quantification of this effect: we map, across model scale and context budget, the regime in which the chunking choice is consequential. We empirically compare TP against FL for action-item extraction and study two questions:

RQ1. Does TP chunking improve task-extraction F1 over FL, and how does the gap depend on chunk size?

RQ2. Is the effect consistent across model scales (7B/8B vs. 120B)?

On a synthetic golden set that mirrors enterprise email, we find that TP significantly outperforms FL in constrained (small-chunk) settings, halving false positives in the best case, with the largest gains on

the small models that are the realistic candidates for on-premises deployment.

2 Related Work

Action-item extraction from email. Extracting actionable tasks from email has been studied for two decades, beginning with rule-based systems and traditional classifiers that identify intent-bearing sentences (Corston-Oliver et al., 2004). With deep learning, encoder models such as BERT were applied to classify sentences as action items, and Mukherjee et al.’s Smart To-Do (Mukherjee et al., 2020) generates to-dos from *single* emails, reporting BLEU and ROUGE scores for single-email to-do generation. These efforts, however, treat each email in isolation. We instead target tasks whose context spans a *multi-email thread*, viewed from the recipient’s perspective—a setting in which a task’s request, modification, and cancellation may be distributed across several messages. Reconstructing such conversation threads is itself a known problem (Dehghani et al., 2013); we take thread structure as given (via metadata) and study how preserving it during chunking affects downstream extraction.

Chunking for LLMs. Chunking is a standard component of retrieval-augmented generation (RAG) and long-document pipelines, and frameworks such as LangChain and LlamaIndex default to fixed-length chunking with optional overlap. Recent work shows that the chunking strategy materially affects RAG quality (Merola and Singh, 2025; Wang et al., 2025): techniques such as late chunking, contextual retrieval, and semantic segmentation aim to preserve global context that naive fixed-size splitting destroys. Such methods, however, are designed for continuous prose and do not model the discrete, metadata-rich structure of email threads, where each message is a self-contained unit with a sender, timestamp, and subject. Our thread-preserving strategy is a domain-specific form of structural chunking for conversational email, analogous to document-aware chunking in the broader RAG literature.

LLMs under tight context budgets. Deploying LLMs in resource-constrained, on-premises settings has drawn growing attention. Quantization methods such as GPTQ (Frantar et al., 2022) reduce model memory costs and let large models run on commodity hardware, but they do not address how

a limited input budget should be allocated across a long email stream; nor do inference-efficiency techniques such as KV-cache optimization and speculative decoding. Our work is complementary: we show that a zero-cost, software-level preprocessing choice—how the input is chunked—can substantially change output quality in exactly these constrained settings.

3 Method

System context. We study the chunking module of an enterprise email-to-task extraction pipeline. In our experiments, emails are loaded from inbox snapshots, filtered by recipient type, capped at a maximum count, *chunked* (our focus), passed to an LLM with a structured-output prompt, and post-processed (validity checks and deduplication). The prompt, post-processing, temperature, and token limits are held identical across strategies; only chunk construction differs.

Thread-Preserving chunking (proposed). TP operates in three steps. (1) *Grouping*: emails are grouped into threads using the Outlook `conversation_id` when present; otherwise the subject line is normalized by stripping common reply/forward prefixes (e.g. RE:, FW:) and used as the key, handling externally forwarded mail that lacks an id. (2) *Formatting*: each thread becomes one text block, emails ordered oldest-first, each prefixed with its timestamp/sender/subject for temporal context. (3) *Assembly*: threads are appended to a chunk until the next thread would exceed the size limit, at which point a new chunk starts. A thread is **never split**; an oversized single thread is kept intact as one over-budget chunk, since splitting it is more harmful than exceeding the limit (Algorithm 1). We therefore treat `chunk_size` as an *operational soft budget* that targets the per-request input size, rather than a hard cap: a single thread longer than the budget is never broken. We quantify and discuss this trade-off in Section 5.

Fixed-Length chunking (baseline). FL concatenates the identically formatted threads into one string and cuts it into consecutive segments of up to `chunk_size` characters, ignoring thread, email, and delimiter boundaries.

4 Experimental Setup

Dataset. For privacy we built a synthetic “Golden Set” of Korean business emails from a

Algorithm 1 Thread-Preserving Chunk Assembly**Require:** formatted threads $T = [t_1, \dots, t_n]$, limit L

```

1:  $C \leftarrow []$ ;  $cur \leftarrow ""$ 
2: for  $t$  in  $T$  do
3:   if  $|cur| + |t| > L$  and  $cur \neq ""$  then
4:     append  $cur$  to  $C$ ;  $cur \leftarrow ""$ 
5:   end if
6:    $cur \leftarrow \text{CONCAT}(cur, t)$  ▷ thread kept whole
7: end for
8: if  $cur \neq ""$  then append  $cur$  to  $C$ 
9: end if
10: return  $C$ 

```

specification defining the JSON schema, challenge types, and labeling rules. It contains **80 cases**, **385 emails**, and **93 ground-truth tasks**; each case is an inbox snapshot of several threads. Challenge tags target chunking-sensitive scenarios: `split_dependency` (request and confirmation separated by unrelated mail), `deadline_update` (a later email revises a deadline), `cancellation` (a task is later canceled), and `missing_conversation_id` (subject-based grouping required). The dataset includes 64 negative annotations describing FYI notices, canceled requests, ambiguous-ownership messages, and other distractors that should yield no task—43 of them emails explicitly flagged as not warranting a task—so a correct system must suppress these; email bodies average 254 characters.

Models and parameters. We use **Qwen2.5-7B-Instruct** (32k ctx; strong Korean), **LLaMA-3.1-8B-Instruct** (128k ctx; different architecture), and **GPT-OSS-120B** (high-capacity baseline). We test six chunk sizes (500/1,000/2,000/3,000/4,000/5,000 chars) with `temperature=0.1` and `max_tokens=4,096`. Each (strategy $\times$ size $\times$ model) condition is repeated **3** times (108 runs); we report mean $\pm$ sample std.

Metrics. We score Precision/Recall/F1 at the task level. A predicted task is matched to the best unmatched gold task using a weighted score: 0.65 times the token-level Jaccard similarity against the gold task and its allowed aliases, plus 0.35 times the overlap of supporting evidence references (the email ids cited as evidence for the task). A predicted task counts as a true positive when this score is at least 0.55; an unmatched prediction is a false positive, and an unmatched gold task is a false negative. Significance uses a paired t -test on the three-run F1 per condition, with paired Cohen’s d as

Model	Chunk	TP F1	FL F1	Δ F1	p
Qwen2.5-7B	500	0.479 $\pm$ 0.017	0.317 $\pm$ 0.016	+0.162	0.002
	1,000	0.474 $\pm$ 0.010	0.356 $\pm$ 0.010	+0.118	0.010
LLaMA-3.1-8B	500	0.497 $\pm$ 0.020	0.374 $\pm$ 0.006	+0.124	0.010
	1,000	0.498 $\pm$ 0.012	0.382 $\pm$ 0.005	+0.116	0.006
GPT-OSS-120B	500	0.428 $\pm$ 0.009	0.359 $\pm$ 0.017	+0.069	0.031
	1,000	0.440 $\pm$ 0.019	0.397 $\pm$ 0.021	+0.042	0.160

Table 1: Mean F1 at small chunk sizes ($N=3$). Bold Δ F1: $p < 0.05$ (paired t -test). Std is sample std.

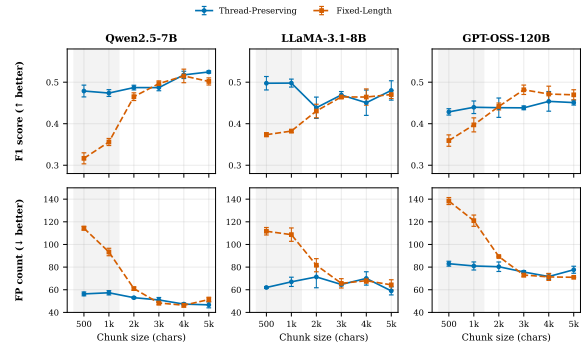


Figure 1: F1 (top, higher better) and false positives (bottom, lower better) vs. chunk size, by model. Blue: Thread-Preserving; orange dashed: Fixed-Length. Error bars: ± 1 SD over 3 runs. Shaded band marks the small-chunk regime ($\leq 1,000$ chars) typical of constrained on-premises deployments, where TP’s advantage concentrates.

effect size; given $N=3$ we read p -values as indicative and emphasize the direction and consistency of the effect. We also count *failed chunks*—chunks for which the LLM call failed or no parseable JSON object could be recovered—which contribute no extractions.

5 Results

TP dominates in constrained settings (RQ1).

Table 1 reports F1 at the two smallest chunk sizes, where TP outperforms FL for all three models, with statistically significant gains in every 7B/8B condition. At 500 chars, Qwen2.5-7B improves from 0.317 to 0.479 F1 (+16.2 pts, $p=0.002$) and LLaMA-3.1-8B by +12.4 pts ($p=0.010$); even GPT-OSS-120B gains +6.9 pts ($p=0.031$). Appendix A lists the individual run scores; in all six of these conditions the lowest TP run exceeded the highest FL run, with margins from 0.003 to 0.126. Figure 1 (top row, shaded region) makes the pattern visual: the FL curve degrades sharply at small sizes and recovers only once chunks grow large enough to contain whole threads.

Model	#chunks		FP		FP $\Delta\%$
	TP	FL	TP	FL	
Qwen2.5-7B	117	344	56.3	114.3	-50.7
LLaMA-3.1-8B	117	344	62.0	111.7	-44.5
GPT-OSS-120B	117	344	83.0	138.3	-40.0

Table 2: False positives and chunk counts at 500 chars ($N=3$). FL produces $\sim 3\times$ more chunks, driving FP up; TP cuts FP by 40–51%.

The driver is fewer false positives. Table 2 attributes the F1 gap to the suppression of false positives. At 500 chars, FL produces 344 chunks on average—nearly $3\times$ the 117 produced by TP—so the model sees multiple fragments of the same thread independently, extracting a task from one fragment while a later fragment that cancels or supersedes it is processed in isolation. Under TP, false positives fall by 40.0–50.7%. Chunk explosion also raises parsing failures, further depressing recall: for LLaMA-3.1-8B, FL yields 46.3 failed chunks vs. 7.3 under TP at 500 chars.

Smaller models benefit more (RQ2). At small chunk sizes, the F1 gain for the 7B/8B models (+11.6 to +16.2 pts, all $p<0.05$) is roughly double that of the 120B model (+4.2 to +6.9 pts): smaller models are less able to infer missing context across fragments, so structurally coherent input matters more—and these are the models used on-premises. At larger chunk sizes the differences between the two strategies are small and inconsistent in sign; given $N=3$ we do not interpret them.

Thread fragmentation also harms evidence traceability. A deployable system must also link each task to the source email(s) that justify it. For LLaMA-3.1-8B we measured how many extracted tasks were correctly attributed to their gold evidence emails, under *exact* (predicted set equals the gold set) and *relaxed* (predicted set contains all core evidence emails) criteria. At 500 chars, TP correctly linked 32.3 tasks (exact) vs. 17.7 for FL—an 83% relative gain—with the same ordering under relaxed (49.3 vs. 39.7). Scattering a task’s supporting emails across chunks prevents the model from seeing them together; this gap narrows with chunk size, like the F1 gap.

Convergence at large chunks. By 5,000 chars a chunk holds an entire case, so FL becomes equivalent to TP (both reduce to 80 chunks, Figure 2); the strategies thus differ meaningfully only in the

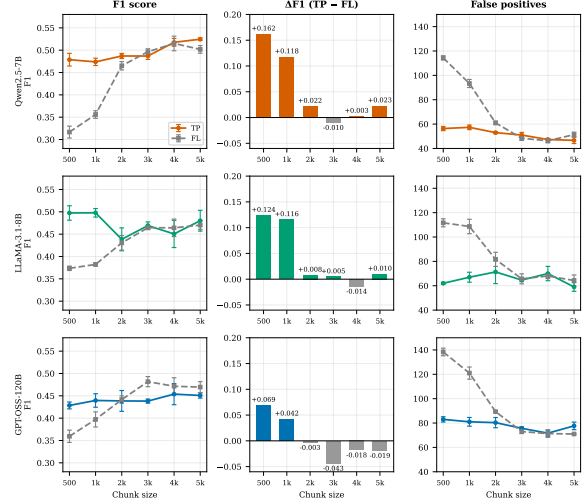


Figure 2: Per-model F1 (left), $\Delta F1 = TP - FL$ (center), and false positives (right) across all six chunk sizes. Rows are models; bars/markers are 3-run means. Positive $\Delta F1$ concentrates at 500–1,000 chars and vanishes at larger sizes.

small-chunk regime imposed by shared-GPU deployments.

Soft vs. hard budgets: a fair-comparison check. Because TP never splits a thread, a thread longer than the budget yields an over-budget chunk—frequent here: at 500 chars most TP chunks exceed the limit (longest over 3,500 chars), while FL respects it by construction. Does TP then win simply by seeing more context? We cannot fully separate the two effects here, though several observations point away from volume alone: its gains come from *fewer, thread-complete* chunks (117 vs. 344 at 500 chars) and concentrate in false-positive and traceability improvements, and they *disappear* once the budget can hold whole threads. Only a hard-capped TP variant—splitting oversized threads at message boundaries while retaining thread headers—would settle the question, and we leave it to future work.

6 Conclusion

For action-item extraction from enterprise email, the chunking strategy is a first-order design decision under a constrained context budget, not an implementation detail. By grouping emails into threads and never splitting one, Thread-Preserving chunking sharply cuts false positives and improves F1 and evidence traceability in the small-chunk regime—most of all on the small models suited to on-premises use—and converges with fixed-length chunking once chunks hold whole threads. It is sim-

ple, zero-cost, and model-agnostic. **Limitations.** Beyond the synthetic, Korean-only data, TP’s soft budget is the key caveat: at small sizes we cannot cleanly separate the gain from thread integrity from that of the larger inputs it admits, which a hard-capped variant would settle.

References

- Rishi Bommasani and 1 others. 2021. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*.
- Simon Corston-Oliver, Eric Ringger, Michael Gamon, and Richard Campbell. 2004. Task-focused summarization of email. In *Text Summarization Branches Out*, pages 43–50.
- Mostafa Dehghani, Azadeh Shakery, Masoud Asadpour, and Arash Koushkestani. 2013. [A learning approach for email conversation thread reconstruction](#). *Journal of Information Science*, 39(6):846–863.
- Elias Frantar, Saleh Ashkboos, Torsten Hoefer, and Dan Alistarh. 2022. GPTQ: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*. Published at ICLR 2023.
- Carlo Merola and Jaspinder Singh. 2025. Reconstructing context: Evaluating advanced chunking strategies for retrieval-augmented generation. *arXiv preprint arXiv:2504.19754*.
- Sudipto Mukherjee, Subhabrata Mukherjee, Marcello Hasegawa, Ahmed Hassan Awadallah, and Ryan White. 2020. [Smart to-do: Automatic generation of to-do items from emails](#). In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8680–8689. Association for Computational Linguistics.
- Zhitong Wang, Cheng Gao, Chaojun Xiao, Yufei Huang, Shuzheng Si, Kangyang Luo, Yuzhuo Bai, Wenhao Li, Tangjian Duan, Chuancheng Lv, Guoshan Lu, Gang Chen, Fanchao Qi, and Maosong Sun. 2025. Document segmentation matters for retrieval-augmented generation. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 8063–8075.

A Per-Run Scores

Table 3 lists the F1 of each individual run for the six small-chunk conditions on which our main claim rests, so that the separation between the two strategies can be judged directly rather than through a test at $N=3$.

Model	Chunk	Str.	Run 1	Run 2	Run 3
Qwen2.5-7B	500	TP	0.459	0.492	0.484
		FL	0.301	0.316	0.333
	1,000	TP	0.462	0.480	0.480
		FL	0.368	0.352	0.348
LLaMA-3.1-8B	500	TP	0.480	0.493	0.519
		FL	0.379	0.367	0.374
	1,000	TP	0.510	0.486	0.498
		FL	0.377	0.385	0.385
GPT-OSS-120B	500	TP	0.439	0.427	0.420
		FL	0.347	0.379	0.352
	1,000	TP	0.423	0.436	0.460
		FL	0.393	0.420	0.379

Table 3: Individual run F1 for the six small-chunk conditions, in execution order.