Got it. I’ll draft a rewritten version of your paper with an **added Related Work section** and **sharper positioning**. I’ll keep your structure intact but tighten the prose and highlight the distinctiveness vs. other approaches. That way, you can cross-compare with your draft and merge what feels strongest.

Here’s a proposed rewrite:

# Intelligent Conversation and Context Management

## Abstract

Large Language Models (LLMs) have transformed how we engage with information and decision-making. Yet their capabilities are constrained by finite context windows, leading to inefficiencies in how conversations, documents, and code are recalled and reused. This paper argues for a shift from *prompt engineering* to *context engineering*. We propose a two-model architecture: a **Specialized Prompt Transformer (SPT)** that learns to generate optimized context windows from effectively infinite conversational history, and a **task LLM** that consumes this window. The SPT classifies, retrieves, summarizes, and interleaves content dynamically, balancing recency, relevance, and domain knowledge. We describe the architecture, training via multi-LLM judgment, and its implications for software engineering and collaborative work.

## Introduction

LLMs are remarkable guessing machines. They reason through conversation—both internal and external—in ways that echo human cognition. Just as humans deliberate between inner voices, LLMs simulate inner dialogues when selecting their next token. This resemblance underscores a deeper truth: conversation is not merely expression, but a *vehicle of thought*.

Documentation has historically memorialized human deliberations, but digital conversation now provides a richer substrate. Unlike static documents, digital dialogues can be fully recorded, indexed, and reused—removing the constraints that forced humans to compress memory into imperfect summaries.

## From Documentation to Conversation

Traditional workflows assume that storing every conversational detail is infeasible. But in AI-augmented environments, entire conversations can be archived and re-queried. This reframes documentation as a derivative artifact of richer conversational history. A system capable of dynamically recalling, compressing, and reinserting conversational fragments could surpass static notes or summaries.

## Rethinking Context

In humans, context is bounded by working memory. In LLMs, context is bounded by token windows. Both are approximations: *how much can I keep in mind to make the next correct guess?*

With digital storage, there is no reason to discard memory. The challenge shifts to intelligently selecting which fragments matter *now*. Current methods—such as compaction or auto-summaries—often fail because they prematurely decide what will be relevant. Human memory, by contrast, retains detail until relevance emerges.

## The Problem with Current Context Management

Today’s LLM sessions waste tokens on irrelevant material (e.g., past code blocks) while missing useful prior insights. Approaches like **Anthropic’s compaction** or **Enhanced-Memory MCP servers** attempt summarization, but they risk discarding nuance or assuming future relevance incorrectly. Users rarely know what will matter later, making pre-summarization brittle.

## Context Windows as System Limitations

The true issue is not *conversation* but *attention*. Both people and LLMs struggle when attention is diffused across irrelevant material. Efficient context management means providing the right subset at the right granularity—*context engineering* rather than prompt engineering.

## The Solution: Dynamic Context Generation

We propose delegating context construction to a dedicated **SPT**, a smaller model trained to optimize context for a task LLM.

* **Short-term memory**: recency-weighted fragments from the current conversation.
* **Long-term memory**: query-retrieved material from infinite archives.
* **Attention scoring**: relevance, recency, and explicit prompt cues.
* **Budgeting policy**: fill ~75% of the window with prioritized content, then interleave retrieved items until the limit.

This yields rolling, high-utility contexts that balance detail with efficiency.

### Example Use Case

A developer asks: *“What was the module we built in June for messaging?”* The system queries past logs, retrieves the “Fogger” module, and reinserts code snippets when relevant. Unlike static summaries, this workflow adapts to the coder’s current focus, minimizing irrelevant context tokens.

## Technical Implementation

1. **Content Classification** – label conversation segments (code, prose, documentation, commands) with value tiers (low, medium, high, boosted).
2. **Content Storage** – archive every fragment in a data lake with metadata.
3. **Content Retrieval** – issue queries conditioned on current input and recency.
4. **Context Generation** – construct draft contexts in reverse chronological order, then interleave retrieved summaries ranked by attention score.

## Training the SPT

### Phase 1 – Domain Pretraining

SPTs are initialized on domain-relevant corpora (docs, codebases, organizational data).

### Phase 2 – Context Optimization via LLM Teaming

A panel of diverse LLMs evaluates candidate contexts against reference answers. By voting on accuracy and utility, the team provides quality signals that tune the SPT’s retrieval and budgeting heuristics. This process optimizes *context construction itself*, not just output generation.

### Production Training

In deployment, the same mechanism continues to refine the SPT with live prompts, balancing quality against compute cost.

## Related Work

**Retrieval-Augmented Generation (RAG)** improves factual grounding by retrieving external documents. Dynamic extensions such as **Self-RAG** add self-critique to decide *when* to retrieve

intelligent\_context\_management\_…

. Our approach differs by optimizing *how retrieved content is budgeted and interleaved* inside the window.

**Virtual memory architectures** like **MemGPT** treat LLM context as paged memory, offloading history and re-inserting summaries

intelligent\_context\_management\_…

. We instead emphasize an SPT that learns explicit budgeting policies and attention scoring.

**Commercial memory features** (ChatGPT, Claude) provide persistent history recall but remain coarse-grained

intelligent\_context\_management\_…

. Our proposal adds systematic classification, query planning, and evaluation to ensure relevance.

**Model Context Protocol (MCP)** enables tool and memory connectors

intelligent\_context\_management\_…

. Our method could act as a “policy layer” atop MCP, orchestrating which fragments to retrieve and at what granularity.

Benchmarks for long-term conversational memory are emerging

intelligent\_context\_management\_…

. Our framework could be evaluated against these by measuring factual recall, temporal grounding, and persona consistency.

## Conclusion and Future Directions

Context windows are not just technical limits but attention bottlenecks. By separating context construction (SPT) from task execution (LLM), we can emulate human-like recall while retaining computational efficiency.

Future research should:

* Benchmark SPTs against long-term dialogue datasets.
* Explore latency/cost tradeoffs of retrieval + interleaving.
* Investigate governance and privacy for persistent conversational stores.

Dynamic context generation reframes the conversation: from prompt engineering toward **learned context engineering**. This promises more faithful recall, better decision-making, and scalable collaboration between humans and AI.

Would you like me to prepare this **rewritten version as a clean .docx or .pdf file** (formatted for easy side-by-side comparison with your original), or just keep it as text so you can manually diff it against your draft?