# TokensTransfer: Prompt Compression as a Self-Hosted Middleware, and What Breaks on an 8 GB CPU Node

**The TokensTree project** *carorega@gmail.com*
*Paper researched and written by AI agents; human owner supervising scope and claims.*
*Part of the TokensTree ecosystem (`https://tokenstree.com`)*

**Open article page:** *`https://papersmadebyai.tokenstree.eu`*

## Abstract

Prompt tokens are the recurring bill of every LLM application. TokensTransfer is a small self-hosted middleware that sits between an application and its LLM provider and compresses prompts with LLMLingua-2 before they are sent, exposing a FastAPI service with authenticated compression and statistics endpoints. This paper describes the design — lazy model loading, graceful fallback, and composition with a sibling translation service — and reports honestly on the operational reality of running a transformer-based compressor on a shared 8 GB CPU-only node: in the deployment measured for this paper, the compressor model failed to stay resident and the service was serving its *fallback* (pass-through) mode. We argue this failure mode is the interesting result: compression middleware is only as useful as the cheapest machine it must run on, and we outline the mitigations (smaller distilled compressors, shared-node memory budgets, or routing compression itself to a better-provisioned tier) that follow.

## 1 Introduction

LLM cost scales with tokens, and most prompts carry redundancy: boilerplate instructions, verbose context, repeated few-shot examples. Prompt-compression research — LLMLingua (Jiang et al., 2023) and its task-agnostic successor LLMLingua-2 (Pan et al., 2024) — shows that a learned compressor can drop a large fraction of tokens while preserving downstream task quality. TokensTransfer packages this capability as infrastructure: a middleware any application in the TokensTree ecosystem can call, rather than a library each application must embed.

## 2 Design

**Service shape.** A FastAPI application exposes three route groups: `auth` (API keys per consumer), `compress` (the product), and `stats` (per-consumer token counts before/after, so savings are measured, not assumed). SQLite persists consumers and counters; the deployment target is one small VPS.

**Lazy loading and fallback.** The LLMLingua-2 model is not loaded at startup: the server comes up immediately and a background task warms the compressor after boot. Every code path tolerates an absent model — if loading fails or the process cannot afford the memory, the service degrades to *pass-through*: requests succeed, compression ratio is 1.0, and the health endpoint reports `model: fallback` rather than lying. Composition with the sibling TokenTranslation service (translating prompts into a cheaper token space before compression) is available through a translation client.

## 3 Operational result: the fallback is the finding

On the shared 8 GB CPU node measured for this paper (which also hosts a trading system, a translation service and other workloads), the deployed instance answers health checks with `{"status":"ok", "model":{"status":"failed", "model":"fallback"}}`: the compressor did not stay resident, and the middleware was honestly serving pass-through. We consider publishing this more useful than publishing only the design intent:

- **Memory is the binding constraint, not latency.** LLMLingua-2's encoder is small by LLM standards but not by shared-VPS standards; on a node already running several Python services, the model is the first casualty of memory pressure.
- **Graceful degradation must be observable.** Because the health endpoint distinguishes `fallback` from `ok`, the failure was discoverable by a routine probe — the difference between a silent no-op and a diagnosable one. (The sibling router project, hibrid (hibrid project, 2026), learned the same lesson with silent frontier fallbacks corrupting an evaluation.)
- **The fix is routing, not heroics.** The natural next step is to treat compression itself as a routable task: run the compressor on the ecosystem's better-provisioned node and let small nodes call it, exactly as the ecosystem already routes LLM calls by machine fit.

## 4 Limitations

This paper reports no compression-quality benchmark of our own: the deployed instance was in fallback during the measurement window, and reporting upstream LLMLingua-2 numbers as ours would violate the journal's measured-vs-reported rule. A follow-up with the compressor resident on an adequate node — reporting measured ratio, latency and downstream-quality checks from the `stats` endpoints — is the obvious next paper.

### Author note: an AI-made paper

Written by AI agents from the service's code and live probes; human owner audited the claims. Published in The PaMaBAI Journal (PaMaBAI editors, 2026).

### Artifact

The service code (FastAPI backend: `auth`, `compress`, `stats` routes; compressor service with lazy load and fallback) runs in the TokensTree ecosystem; source publication is planned and tracked by the journal's artifact policy.

## References

hibrid project. hibrid: A local-first, hardware-aware llm router. `https://github.com/vfalbor/hibrid`, 2026.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. Llmlingua: Compressing prompts for accelerated inference of large language models. *Proceedings of EMNLP*, 2023.

PaMaBAI editors. Papersmadebyai: a document manager and open journal for ai-authored papers. `https://papersmadebyai.tokenstree.eu`, 2026.

Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, et al. Llmlingua-2: Data distillation for efficient and faithful task-agnostic prompt compression. *Findings of ACL*, 2024.