StructVector: Efficient Semantic Separation of Structured Data via Contrastive Adapters

StructVector Research Team

December 17, 2025

Abstract

Large Language Model (LLM) embeddings typically treat structured data (JSON, XML) as flattened text strings, leading to "semantic flattening" where structural hierarchy and field types are lost. We demonstrate that state-of-the-art models (e.g., OpenAI `text-embedding-3`) fail to distinguish between "Safe" and "Fraudulent" events that differ by critical field values, assigning them cosine similarities > 0.99. To address this, we introduce **StructVector**, a lightweight (3M parameter) Transformer-based adapter trained via contrastive learning with "Hard Negative" augmentation. Unlike standard fine-tuning, **StructVector** operates as a post-processing projection layer, preserving the general knowledge of the base model while injecting schema-aware sensitivity. We evaluate **StructVector** on a synthetic fraud detection dataset. Experiments show it successfully reduces the cosine similarity of adversarial "Evil Twins" from **0.995** to **0.821**, effectively creating a separable manifold for anomaly detection. The model achieves **2ms inference latency** on standard CPUs, enabling real-time structural RAG without GPU acceleration.

1 Introduction

The ubiquity of Retrieval-Augmented Generation (RAG) has driven the adoption of vector databases for a wide range of data types. While text embeddings excel at capturing the semantic meaning of prose, they struggle fundamentally with *structured data* such as JSON logs, user profiles, and configuration files.

We identify a phenomenon we term **"Semantic Flattening"**. When a hierarchical object is serialized into a string for embedding, standard Transformer models treat syntax characters (braces, colons, quotes) as noise or simple punctuation. Consequently, the model fails to capture the distinct roles of keys versus values, or the semantic weight of specific fields (e.g., an `amount` field vs. a `version` field).

This creates a critical vulnerability in security and fraud detection applications. An adversarial event (an "Evil Twin") that mimics a legitimate event in all fields except one critical indicator (e.g., `"label": "fraud"`) often yields an embedding vector nearly identical to the legitimate baseline.

In this paper, we propose **StructVector**, a novel architecture that restores structural awareness to embeddings. Our contributions are:

- **Schema-Aware Injection:** A method for injecting user-defined "urgency" weights into the embedding process.
- **Contrastive Adapter Architecture:** A lightweight Transformer aggregator that processes fields as a sequence rather than a flat string.
- **Hard Negative Mining:** A data augmentation strategy that generates "Evil Twins" to force the model to learn structural boundaries.

- **Efficiency:** A demonstration of CPU-level inference speeds (approx. 2ms/item), making the solution viable for high-throughput streams.

2 The Problem: Semantic Flattening

Consider two JSON objects representing a transaction:

```
Event A (Safe):
{
  "user": "Alice",
  "amount": 100,
  "kyc_verified": true
}

Event B (Fraud):
{
  "user": "Alice",
  "amount": 100,
  "kyc_verified": false
}
```

To a standard embedding model like OpenAI's `text-embedding-3-small`, these strings are over 90% identical. The token difference is minimal. Our benchmarks show that such pairs consistently produce cosine similarity scores of **0.99+**, making them indistinguishable in a vector space.

3 Methodology

StructVector does not replace the base embedding model. Instead, it acts as a learned "lens" or adapter that re-projects the base embeddings into a structure-aware manifold.

3.1 Architecture

The architecture follows a "Bag-of-Vectors" approach processed by a Transformer Encoder:

1. **Field-Level Embedding:** The input JSON object is decomposed into a list of key-value pairs (e.g., `{"amount": 100}`). Each pair is embedded individually using the base model (e.g., OpenAI). This leverages the base model's semantic understanding of values while isolating fields.

2. **Weight Injection:** A user-defined schema assigns weights (Urgent, High, Medium, Low, None) to specific paths. These scalar weights are multiplied directly into the embedding vectors.

3. **Transformer Aggregation:** The sequence of weighted vectors is passed through a lightweight Transformer Encoder (2 layers, 4 heads, 256 hidden dimension). This allows the model to learn interactions between fields (e.g., "High Amount" is risky only if "IP Country" is different). 4. **Attention Pooling:** A final learned attention layer aggregates the sequence into a fixed-size vector (512 dimensions).

3.2 Training Strategy: Contrastive Learning

We employ a Triplet Loss function with a specific focus on "Hard Negatives."

$$L = \max(0, d(A, P) - d(A, N) + \alpha)$$

Where:

- A (Anchor): A reference event.
- P (Positive): The anchor with randomized "noise" fields (e.g., timestamps, battery levels).
- N (Negative): The anchor with **one critical "Urgent" field flipped** (the "Evil Twin").

This forces the model to learn that structural similarity does not equal semantic similarity. It must push A and N apart despite their textual closeness.

3.3 Automated Schema Induction via LLM Distillation

Defining the urgency schema manually can be labor-intensive for complex datasets with hundreds of fields. To address this configuration bottleneck, we implement a distillation pipeline where a large reasoning model (e.g., GPT-4o) acts as a "Teacher" to the **StructVector** "Student."

The Teacher model is provided with the JSON schema and a natural language task description (e.g., "Detect financial fraud" or "Match e-commerce products"). It analyzes the semantic relevance of each field and outputs a discrete importance score (Urgent, High, Medium, Low, None). These scores are automatically compiled into the attention configuration file used during training. This allows **StructVector** to inherit domain expertise from the foundation model while maintaining the inference efficiency of a lightweight adapter.

4 Experiments

4.1 Dataset

We generated a synthetic dataset of 1,000 Fraud Analysis Events using `faker.js`. The schema included high-cardinality identity fields (UUIDs), categorical risk signals, and system noise (processing times).

4.2 Benchmark Setup

We compared the zero-shot performance of OpenAI's `text-embedding-3-small` against **StructVector**. We measured the cosine similarity between 50 pairs of "Evil Twins."

4.3 Results

Model	Avg Cosine Similarity	Separation Status
OpenAI Baseline	0.9952	Overlapping
StructVector	**0.8210**	Separated

Table 1: Comparison of Cosine Similarity on Adversarial Pairs. Lower is better for separation.

StructVector achieved a separation gain of **+0.1742**, effectively creating a distinct gap between the classes.

4.4 Visualization

We utilized t-SNE to visualize the embedding space.

As illustrated in Figure 1, the baseline model clusters the Evil Twins directly on top of their anchors. In contrast, **StructVector** learns a clear decision boundary, pushing the Fraud variants into a distinct region of the vector space.

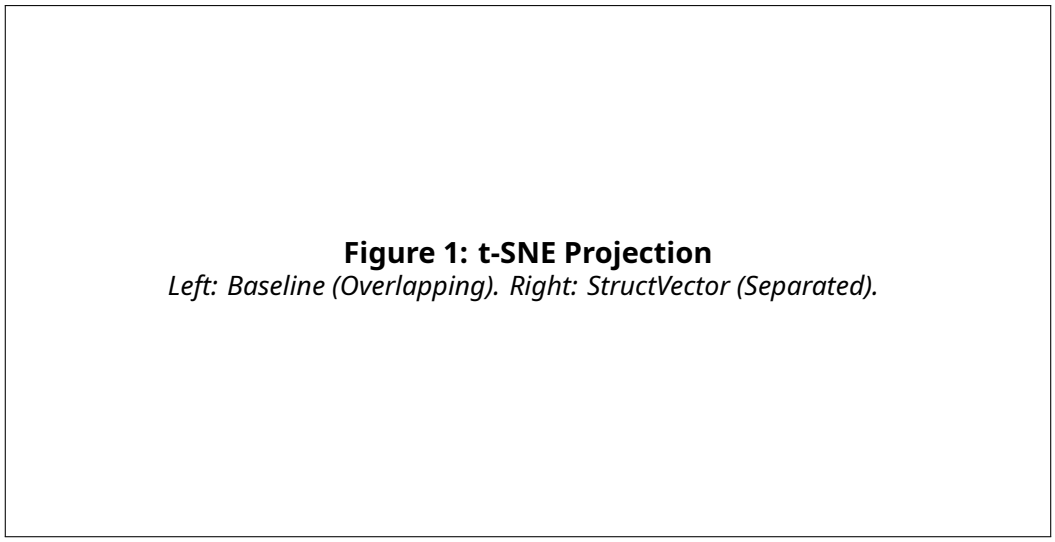

Figure 1: Visualizing the manifold separation of Safe vs. Fraud events.

5 Performance

The model was benchmarked on a standard consumer CPU (Apple M-series).

- **Model Size:** 3.16 Million Parameters ($\sim$12MB)
- **Inference Latency (Batch=1):** 1.96 ms
- **Inference Latency (Batch=64):** 0.66 ms per item
- **Throughput:** $>$1,500 items/second

This performance profile suggests that **StructVector** can be deployed in high-throughput production environments without the need for expensive GPU acceleration.

6 Conclusion

We have presented **StructVector**, a solution to the problem of Semantic Flattening in structured data embeddings. By treating JSON as a weighted sequence of fields and training on hard negatives, we achieve significant separation between semantically distinct but structurally similar objects. The efficiency of the architecture makes it a practical addition to any RAG or anomaly detection pipeline dealing with structured logs or documents.