Hyperflow v0.1

Version 1.o - MVP

Powered Intelligence

—

A Artificial Intelligence Architecture written and implemented from scratch in vanilla Go language, C++ and JavaScript.

Explains how to make current Transformers 100x more faster, efficient and powerful with limited resources.

Author: Pawan Yadav.

About

What is HDFN?

The **HDFN-1** is an artificial intelligence based on **HyperFlow** architecture which is itself based on the theory of *Hierarchical Dynamic Flow Networks*(HDFN).

Development under extreme resources.

A machine learning model that i am building on a Pentium G3320 and 4GB RAM.

Resource Consumption

According to my plans and theories it can work on 60% less memory because of it's **Compressed State Management** where it manages to compress it's state in the memory and runs fast because HyperFlow projects a computational complexity of `O(n·log(n)·d)` , a substantial improvement over the Transformer's `O(n^2d)` for sequence length *n* and dimension *d*. HyperFlow predicts a 3-5x faster initial training speed and a 2-4x faster generation (inference) speed. The hierarchical learning and adaptive sparsity mechanisms are expected
to enable the model to learn effectively from 30-50% less data. Multi-format capability: The architecture is designed with native support for diverse
output formats, including text, code, and documents, offering a single model solution for varied tasks.

Example:
A 1,000,000 Seq model taking 24 Core CPU, 64GB RAM and 100GB dataset.
Will take:

Resource	Baseline (Transformer)	HyperFlow (Avg Estimate)
CPU Cores	24	**~6 cores**
RAM	64 GB	**~25–26 GB**
Dataset	100 GB	**~60 GB**

If the optimization is: ~60% saving of Memory, ~40% minus of dataset and at 4x Efficency of Inference/Training.

📊 Breakdown:

- **CPU:** Since it is ~4× faster, the hyperflow only needs 1/4 the compute for the same throughput → ~6 cores.

- **RAM:** 40% of 64 GB = 25.6 GB → rounded ~26 GB average RAM required.
- **Data:** Learning from 40% less data = 100 GB × 0.6 = **60 GB dataset**.

Architecture

The current development is under progress.

Tokenizer

Currently the tokenizer i am using is "SentencePiece" with BPE learning fallback.
Tokenizer is named after my brand name `Algoritms.ai` .
The Tokenizer's name is **`Algoritms A1013.`** A subword sentencepiece tokenizer with these specs:

1. 16,000 Vocabulary
2. 1.1GiB (JSONL) 1013MB (Trained) corpus.

Trained using **`sentencepiece (spm_train and spm_encode)`** from Google's Github repo (github.com/google/sentencepiece/) I installed and compiled it from scratch using G++ and CMake.

Dataset

The dataset i am currently using is:
[Writing Prompts](), [OpenbookQA](), [RACE](), [SciQ](), [SQuAD]() and my Custom handwritten small corpus with usual commands and instructions ("You are Algoritms HDFN1 a friendly chatbot", etc).

Algoritms Training Interface

A custom AI/ML/DL framework written in Go language and raw math without using a single library. The framework is also protected by a EULA which allows personal custom tuning.
Currently it only has what i want instead of putting everything in stripping of all the useless bloating stuff.
Like:

1. Hidden Markov Models
2. Greedy, Top-p, Top-K and beam sampling
3. Linear Classifing

And constantly adding my custom HyperFlow, HDFN and Tokenizing.
PROOF it's working or not?
Answer: Currently it did 97% accurate POS-Tagging with only 35KB of DataSet.
Under 100KB it does solid POS Tagging.

Containing: 6.3KB - Go language source code (hmm.go)
35KB Corpus and finally when ran the hmm it outputs a 35KB .JSON model
Which runs in a second but still gives solid outputs.

Model

The HDFN's development is currently using the Autoregressive architecture (Also used by **Qwen**).
Example how it works:

```
const tokens = tokenizer("Who are you?");
// [1380 166 63 13983] or [_Who _are _you ?]
const embeds = embed(tokens);
/*
1380  → [0.4545, 0.3434, 0.3432, 0.5623]
166   → [0.2342, 0.5999, 0.3457, 0.5768]
63    → [0.2348, 0.5776, 0.4767, 0.4567]
13983 → [0.4566, 0.6576, 0.5765, 0.4868]
*/
// Here every word explains something about token example: 1380's
0.4545 can mean "QUESTION", 0.3434 can mean "
```

Embedding

The HyperFlow introduces ***Static Intent Logic Seed*** (SILS). In the Training process. Where instead of randomly assigning the embeddings at random places the SILS Place **Logic** and **Intent** at **Static** places and it is so reliable you can put it in any SEED embedding process (Where you don't have any seed embedding already).
Like how SentencePiece doesn't require a preprocessing, SILS also doesn't require.
I am explaining this with a simple example:
INPUT = "Woman"
TARGET = ["Human", "Female", "Girl", "Mom", "Sister"]

```
Dataset
"He ran to his mom"
"Her sister was kind"
"The girl played chess"
```

Here the model learns that the Woman is a human being:
Output = ["Human"]
The model predicts that the "Woman" is "Human" and is a "Female".
Output = ["Human", "Female"]
... (processing)

Final output comes:
Output = [“Human”, “Female”, “Girl”, “Mom”, “Sister”]

To minimize the loss the SILS should use a reliable learning method like **Reinforcement Learning** where if the model predicts wrong reevaluate the model else if it is correct increase the confidence for the model.

Initially the model itself can’t find a UNDERSTANDING like “Woman ≈ Human”
So the SILS requires a backend like GloVE / Word2Vec to find the embedding understanding and safely and accurately assign the vector.

Short Explaination in 5 Lines:

1. **SILS** assigns structured values to embedding positions:
   `emb[0] = GENDER`, `emb[1] = SPECIES`, `emb[2+] = SIMILAR`.
2. It uses **logical and semantic intent** to seed embeddings instead of full-random floats.
3. It fetches **starter similarity** from external models like Word2Vec/GloVe to anchor meaning.
4. It positions similar concepts close (e.g. `"Girl"` → `"Mom"`, `"Sister"`) + assigns core identity tags.

Once seeded, it feeds into the main training pipeline with high consistency + easier correction