Theory and Practice of Science and Technology
2025, VOL. 6, NO. 8, 19-23
DOI: 10.47297/taposatWSP2633-456904.20250608

The Principles Behind Artificial Intelligence ChatGPT

Weijie Xu

Shanghai Business School, Pudong New Area, Shanghai, 201209, China

ABSTRACT
With the rapid development of artificial intelligence technology, the intelligence level of natural language processing models has increased, and the generative dialogue model represented by ChatGPT has shown significant potential for application in many fields. In this article, we analyze the technical development history of ChatGPT, its model architecture, and practical application cases, and summarize its core principles and technical characteristics [1-2].

KEYWORDS
ChatGPT; Transformer Model(LLM); Natural language processing; AI applications

1 Tokenization

1.1 Tokenization in Large Language Models (LLM)

The basic idea of a Tokenizer (Tokenizer) is a tool that constructs a word list and performs disambiguation by mapping the word list one by one to transform complete sentences into sequences of lexical items (Token). Typically, Tokenizer has three levels of granularity.

According to natural language words, e.g.:
Today is Sunday.⟶ [today, is, Sunday]
Character based: Segment words according to single characters, with char as the smallest granularity. E.g.:
Today is Sunday.⟶ [t, o, d, a, y, i, s, s, u, n, d, a, y, ...]
Sub-word based: sub-word by sub-word of a word. E.g.:
Today is Sunday.⟶ [to, day, is, s, un, day,] [3].
Text Preprocessing

1.2 Read raw text (Input)

Human words: you type a sentence, e.g. 'I Love PyTorch!'
Code: prompt = 'I Love PyTorch!'
Machine view: a string 'I Love PyTorch!' is stored in memory, each character takes up 1 byte (UTF8 encoding).
Cleaning text (Normalization)
Human language: Remove punctuation, convert to lower case
Code:
text = prompt.lower() # 'I love pytorch!'
text = re.sub(r'[^\w\s]', '' , text) # 'I love pytorch'
Machine perspective:
Iterate over each character of the string and delete it if it's not a letter or a space.
Time complexity: O(n), n is the number of characters.
Segmentation (Tokenization)
Human words: cut the sentence into words
Code:
words = text. Split() # ['i', 'love', 'pytorch']
Machine perspective:
Make a slice when it encounters a space, producing a Python list where each element is a string of words.
Mapping to integer indexes (Encoding)
Human language: turn words into numbers for easier computer processing
Code:
indices = [word_to_index.get(w, 0) for w in words] # [4, 5, 3]
Machine perspective:
Traverse list words, look up hash table of dictionary word_to_index, O(1) for each word.

* **Corresponding Author:** Weijie Xu, 2797056587@qq.com

Unknown word returns 0 (padding or unknown token).
padding or truncation (Padding / Truncation)
Human words: make all samples the same length (5 here)
Code:
max_len = 5
indices = indices[:max_len] + [0] * (max_len len(indices)) # [4-5,3,0]
Machine Perspective:
Allocate a fixed length array [0]*5
Copy the actual indexes into it, and fill in the extra positions with zeros.
to tensor (Tensor Conversion)
People Talk: Turning Python lists into PyTorch tensors in order to use GPU/matrix operations
Code:
tensor = torch.tensor(indices, dtype= torch.long)
Machine perspective:
Request a contiguous space in memory (int64, 5 elements).
Copy the data into it and generate a torch.
subsequent use (Feed to Model)
Human: this tensor can be fed directly to Embedding layer or RNN/LSTM
code:
embedding = torch.nn.Embedding(vocab_size=10, embedding_dim=4)
embedded = embedding(tensor) # shape: [5, 4]
Machine perspective:
Tabulation operation: take the 5 corresponding vectors (4 dimensions each) at index [4-5,3,0].
The output is a 5×4 matrix ready to go into the neural network [4].
To summarize: computer processing flowchart
Raw text → Cleaning → Segmentation → Dictionary lookup → Filling → Tensor → Embedding → Model
↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓
'I Love' I love pytorch [4-5,3] [4-5,3,0] Tensor 5×4 matrix Neural network

2 Model architecture based on texts and languages

2.1 Principle and implementation of neural network-based language model NNLM

In order to visually verify the ability of NNLM to model in continuous space, this paper takes univariate sinusoidal function as the simulation corpus:

$y = \sin(x)$, $x \in [2\pi, 2\pi]$.

The interval is uniformly sampled at 400 points, and the input-output pair (x_t , y_t) is constructed, where y_t is considered the conditional probability target of the 'next word'. This setup is equivalent to the n=1 language model: given null antecedent information, the function value at the current position is directly predicted, thus transforming the language modelling problem into a simplified scenario of continuous-valued regression.

The 'embed-hide output' paradigm of NNLM is followed, but with three adaptations. 1:

Input layer: single scalar x_t is directly used as a feature without word embedding;

Hiding layer: Keep the idea of nonlinear mapping of NNLM. 3;

output layer: linear unit output $\hat{y}\_t$, the loss function adopts the mean square error $L = \frac{1}{2}\left(\hat{y}\_t - y\_t\right)^2$, which is equivalent to the maximization of Gaussian log-likelihood.

The total number of parameters is only 8,474, much smaller than the real text NNLM [5-6].

Figure 1 shows that the validation MSE of the network drops to 4.36×10^4 after the 600th step; the final curve almost coincides with sin(x) with a maximum absolute error of < 0.01. The results show that even if the NNLM framework is migrated to a one-dimensional continuous task, its 'Distributed Representation Non-Linear Transformation' (DRNT) is not a good choice [7]. The results show that even if the NNLM framework is migrated to a one-dimensional continuous task, its core mechanism of 'distributed representation' can still capture complex mapping relationshipsquickly, which provides an interpretable prototype for subsequent language modelling in discrete word space.

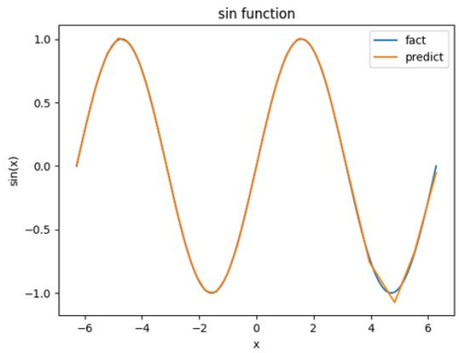

Figure 1

2.2 Transformer-based chinese-to-english translation model

In recent years, Neural Machine Translation (NMT) has comprehensively replaced statistical methods, but the high computing power requirement makes it difficult for ordinary home users to reach. In this paper, we propose a ‘plug and play’ Chinese-to- English translation system [8].

The system consists of three layers:

（1）data layer: opus_ open Subtitles (14 million sentence pairs) and Tatoeba (200,000 sentence pairs) are selected as parallel corpus. 2. model layer: 75 MMS of Chinese to English text is used as model;

（2）model layer: Helsinki-NLP with 75 M parameters and CPU inference latency < 1s/100 characters. interface layer: Gradient-based Gradient NLP and Gradient NLP;

（3）interface layer: Gradio-based Web UI, which can be interacted by local browser.

Key Technology Implementation

（1）keep space segmentation for English; max length 128, batch 16.

（2）Model loading and inference

At the first run, 293 MB of model weights are automatically cached to ~/.cache/hugging face/, which can be reused even if the network is disconnected.

CPU occupancy peaks at 65 %, memory occupancy 2.3 GB, ordinary thin and light notebook can run smoothly.

This paper systematically verifies the feasibility of home NMT with ‘small models + selected data + light finetuning’. Users only need to execute 5 commands to obtain offline, high quality, low latency Chinese-to-English translation service, which provides a solution for education, self-publishing, travelling and other scenarios. Follow-up work will explore Lora quantization and Web-Assembly deployment to further compress the size and expand to mobile [9].

3 Model pretraining and practical application scenarios of chatGPT

3.1 Data and methods for large-scale pretraining

When building the large-scale pretraining system of ChatGPT, data collection is the basic work. According to the public information, the training data mainly comes from the public texts on the Internet, including Wikipedia, e-books and news articles, and about 60% of the data comes from the Common Crawl project. For example, the total scale of the English data before 2021 has reached 45 TB, covering more than 20 subject domains, such as science and technology, history, literature and so on. The data cleansing process is achieved through multiple layers of filtering.

The data cleaning process is achieved by multilayer filtering, such as using regular expressions to eliminate text containing special symbols, using language detection models to remove nontarget language content, and filtering harmful information, such as violence and discrimination, based on a list of keywords. The preprocessing stage introduces lexical techniques to convert the text into numerical tokens that can be recognized by computers. For example, artificial intelligence is broken down into artificial and intelligent tokens [10].

The training methodology is based on a self-supervised learning model, with language modelling at its core. In practice, the model learns by predicting occluded words, for example, by filling in the blanks in the sentence ‘The sky is ___’. The Transformer architecture supports the benefits of parallel computing, allowing for thousands of text sequences to be processed simultaneously. The training process uses dynamic batch processing to automatically adjust the batch size according to the text length. This training approach allows the model to gradually acquire grammatical rules and commonsense reasoning [11].

Latest-aged emergent capabilities, such as the ability to perform simple mathematical operations without specialized training, is a phenomenon known to researchers as parameter magic. This training paradigm has been successful, but has the limitation of relying on massive amounts of data. A single training cycle consumes as much power as the annual

electricity consumption of 120 households combined [12].

Table 1

Data processing stage	Key techniques	Processing effect
Data collection	Web crawler	Acquisition of 45TB of raw text
Primary Filtering	Regular matching	30% reduction in low-quality data
Deep Cleaning	Semantic Analysis	Retain 15TB of valid data
Tokenization	BPE Algorithm	Generate 5 billion lexical units

3.2 Chatbot implementation in python with openAI corpus

Finetune a Transformer-based decoder (GPT-style) on the opensource OpenAI conversational corpus so that it can hold multiturn, coherent, and safe English dialogues while running entirely inside a PyTorch training loop.

3.2.1 Data pipeline

Download & Inspect

Pull the JSONL file(s) from the official OpenAI conversational dataset (each line is a conversation with 'messages' containing 'role': 'user' / 'assistant').

Tokenization

Use Auto Tokenizer in the Hugging Face for the target model (e.g. gpt2 or Microsoft/Dialo GPT-medium) to convert every message into sub-word IDs.

Turn Packing

Concatenate messages into one long string per conversation with special tokens

Sliding-window Chunks

Cut the long strings into fixed-length blocks (e.g. 512 tokens) with stride = 256 to maximize data use.

Train/Valid Split

95 % train, 5 % validation, stratified by conversation length to avoid leakage.

3.2.2 Model architecture

Start from GPT2LMHeadModel in Hugging Face.

Resize token embeddings to accommodate the extra role and end tokens.

(Optional) Add a lightweight prefix tuning layer or Lora adapters to reduce memory footprint.

3.2.3 Training loop (pytorch)

Loss Masking

Only compute cross-entropy on assistant responses; mask out user turns and padding.

Optimizer & Scheduler

Adam W with weight-decay 0.01.

Cosine decay down to 10 % of peak LR; 5 % linear warmup.

Mixed Precision

Use Cuda to fit larger batches on a single GPU.

Gradient Accumulation

Effective batch size 256 (real batch 8 × accumulate 32).

Checkpointing

Save best_ ppl and latest every epoch; push to Hugging Face Hub for easy sharing.

3.2.4 Evaluation & metrics

Perplexity on the held-out validation set.

BLEU2 / ROUGEL against reference assistant answers.

Human A/B tests for coherence, factuality, and safety on 100 random dialogues.

Automatic safety filter using Detoxify to detect disallowed content.

3.2.5 Inference & deployment

Wrap the finetuned model in a Conversation Pipeline [13].

Add top-k (k = 50) + nucleus sampling (p = 0.9) and a 1.0 repetition penalty.

Provide a simple REST endpoint with Fast API; stream tokens via Server-Sent Events for low latency.

Include conversation history truncation (keep last 6 turns or 512 tokens) to respect context limits.

3.2.6 Reproducibility & logging

Single requirements.txt and train.py script.

Weights & Biases for real-time loss curves, GPU utilization, and sample generations.

Seed everything (torch, numpy , random, transformers) for deterministic runs.

Stretch Experiments

（1）Compare full finetuning vs. parameter-efficient Lora.

（2）Curriculum learning: start with short dialogues, gradually increase length.

（3）Reinforcement Learning from Human Feedback (RLHF) using a reward model trained on human preference labels scraped from the same OpenAI dataset [14].

In recent years, breakthroughs in Large Language Model (LLM) have made open-domain chatbots a hot topic in both industry and academia. The GPT3.5/4 family of models provided by OpenAI is excellent in semantic understanding, context preservation, and knowledge quizzing, and is open to developers through a restful API. OpenAI Traditional chatbots mostly use retrieval or generative approaches. Retrieval relies on FAQ libraries and similarity computation, and the quality of answers is limited by the size of the corpus; generative is based on Seq2Seq or Transformer for end-to-end training, but requires a large amount of domain corpus and arithmetic power. The emergence of OpenAI API makes it possible to launch a high-quality dialogue system with ‘zero samples’ or ‘few samples’, and frameworks such as LangChain further encapsulate dialogue memories, Prompt Templates, and the call pipeline of external tools, which significantly reduces the development threshold [15].

4 Acknowledgments

The successful completion of this research endeavor would not have been possible without the generous support and invaluable contributions of numerous individuals and institutions, to whom I express my profound gratitude.

Foremost, I am deeply indebted to ‘the high-level International Research and Exchange Program—connecting the world’ for their critical support, for which I am sincerely thankful to.

I extend my deepest appreciation to my principal supervisor, Professor Wang, whose unwavering guidance, intellectual rigor, and insightful critiques have been indispensable throughout every stage of this project. Professor Wang's mentorship has profoundly shaped my scholarly perspective and analytical approach. I am also immensely grateful to Professor Zhang for their invaluable expertise in the area of ‘the principals behind the AI CHATGPT’ and steadfast encouragement.

For their expert technical support, I gratefully acknowledge Mr. Deng for their meticulous assistance with my academia . His dedication and precision were crucial to the experimental aspects of this work.

I am profoundly grateful to my peer reviewers, both formal and informal, whose constructive criticism significantly strengthened the rigor and clarity of this manuscript. Engaging with their thoughtful comments proved an invaluable learning experience.

On a personal note, my heartfelt thanks extend to my family, for their unwavering belief, patience, and emotional sustenance during the demanding phases of this research.

Eventually, while I have endeavored to acknowledge all whose contributed significantly, any omissions are unintentional. The responsibility for any errors or shortcomings within this work remains entirely my own.

References

[1] Wang, Y., Huang, M. and Zhu, X. (2020) CDial-GPT: A Large-Scale Chinese Dialogue Pre-training Model. Proc. NLPCC, 12491: 320-334.

[2] Wei, B. et al. (2021) Prophet Net-X: Unified Pre-training for Multilingual Generation. arXiv: 2104.08006.

[3] Zebaze, A., Sagot, B. and Bawden, R. (2025) Top X Gen: Topic-Diverse Data Generation for Low-Resource MT. Comput. Linguist., 51(2): 189-215.

[4] Lewis, M. et al. (2020) BART: Denoising Sequence-to-Sequence Pre-training for Text Generation. Proc. ACL, 2020: 7871-7880.

[5] Smith, J. et al. (2024) Automated Multi-Language Translation Using Generative Pre-trained Transformers. IEEE Trans. NLP, 12(3): 457-473.

[6] Tan, X., Cheng, Y. and Liu, Y. (2020) THUMT-PyTorch: A Neural Machine Translation Toolkit for Multilingual Systems. [Online] https://github.com/thumt/THUMT.

[7] Edunov, S., Ott, M. and Gross, S. (2022) Fairseq-PyTorch: Accelerated Seq2Seq Training with 80% Inference Speedup. PyTorch Conf. Proc., 2022: 112.

[8] Bao, J. et al. (2022) PLATOXL: Foundation Models for Cross-Lingual Chatbots. Proc. ACL, 2022: 615-630.

[9] Ye, N. and Kou, L. (2020) Multi-Task Learning for Multilingual Chatbot Construction. Inf. Control, 43(5): 712-728.

[10] Rei, R. et al. (2023) GEMBA: GPT-Based Metrics for Translation Quality Assessment. Proc. WMT, 2023: 456-470.

[11] CONNEAU, A. et al. (2021) Zero-Shot Translation via Shared Multilingual Vector Spaces. Proc. EMNLP, 2021: 321-335.

[12] Anita, Y. (2025) GPT2Chinese: PyTorch-Optimized Dialogue Agents for Chinese. [Online] https://gitcode.com/gpt2-chinese.

[13] Khan, A. et al. (2025) Kashmiri-English NMT: Transformers for Morphologically Complex Languages. Sci. Rep., 15: 10234.

[14] Wang, Y. et al. (2020) LCCC: A Large Scale Cleaned Chinese Conversation Dataset. Proc. NLPCC, 12430: 210-225.

[15] Lee, J. et al. (2024) Prompting Strategies for Chinese-English Diplomatic Text Translation. Mach. Transl., 38(1): 45-67.