GOSIM

GOSIM AI Paris 2025

Beyond Next Token Prediction in Code LLMs

Diégo Rojas, Zhipu.AI

GOSIM AI Paris 2025

Introduction

GOSIM

Next Token Prediction

Autoregressive Generation

At each time step, predict the next token based on the previous tokens.

$$P(X) = \prod_{t=1}^{T} P(x_t | x_{<t})$$

Subword Tokenization

Tokens are words or subwords.

N	ext	_	Token	_	Pred	iction

① Autoregression in LLMs

② Tokenization in LLMs

③ Promising Directions (& Future Plans)

④ Announcement

Self-Introduction

GOSIM

I'm **Diégo**, 22 years old, Research Engineer, based in Paris.

Studied at **EPITECH** (2019-2023), **Tsinghua University** (2022-2024).

Working on Code LLMs at **Zhipu.AI** (1.5 years)

GLM 130B, ChatGLM, CogView, CogAgent, WebGLM, CogVideo(X), CodeGeeX, GLM Z1

CodeGeeX

GOSIM

We've built a cool product around CodeGeeX, launched in 2023, now **1M+ users**, **100K+ DAU**.

The main features are **code completion**, **codebase search**, **agent**, **code interpreter**, **edit prediction**, etc.

billions of tokens/day

We're motivated to explore the design space for code LLMs.
Going beyond NTP is a promising direction.

Why Focus on Code?

While similar to natural language, code has some unique characteristics.

- **Structured** by nature
- Often has **lower entropy** than natural language (repetition, syntax)
- Widely different **entropy distribution** compared to natural language
- Wide range of **coding tasks** with varying speed/performance requirements

Code is a great starting point to optimize.

NL

```
The quick brown fox jumps over the lazy dog.
```

Code

```
def fibonacci(n):
    return n if n < 2 else fibonacci(n-1) + fibonacci(n-2)
```

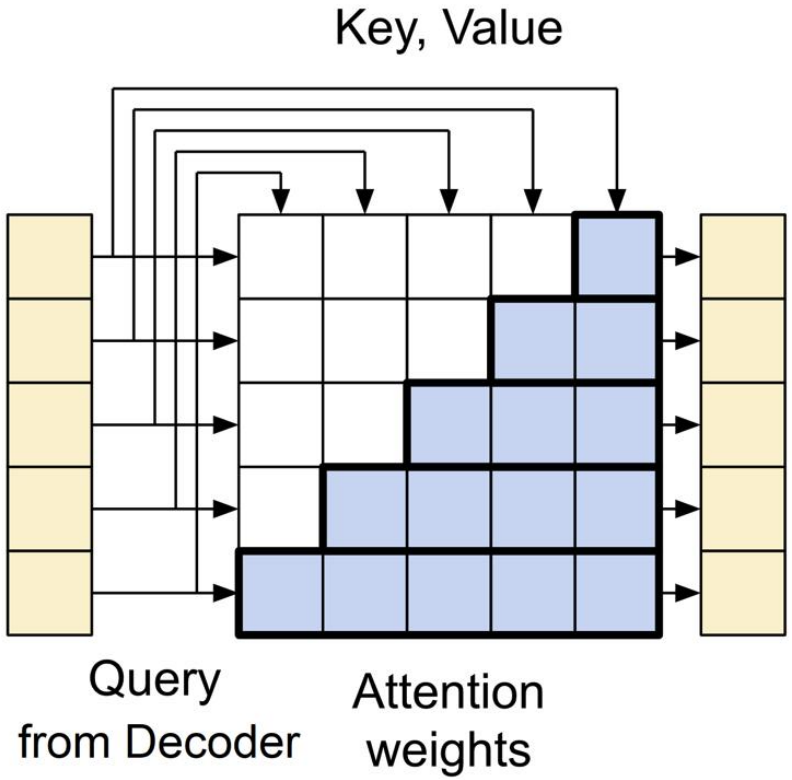
GOSIM

GOSIM AI Paris 2025

Autoregression in LLMs

Efficient Training, Inference

Autoregressors are easy and efficient to train. KV-cache speeds up inference.

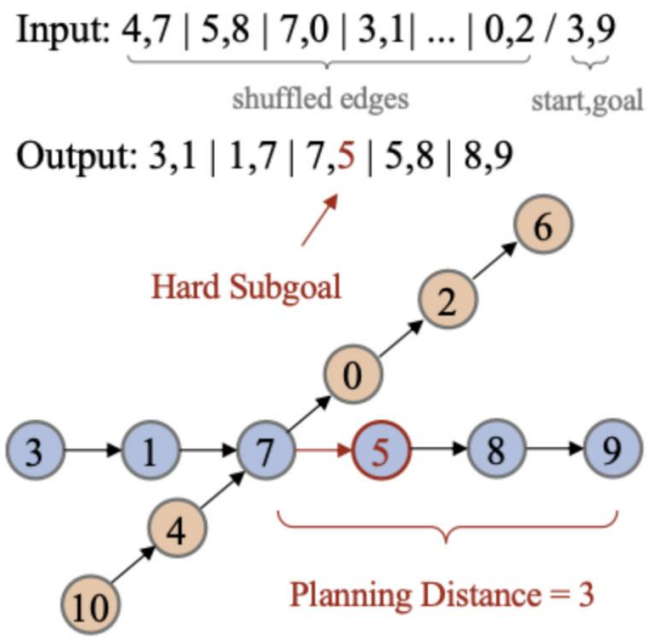


- During training, makes a prediction for **all tokens**.
- During inference, leverages **activation caching**.

Planning Ability, Speed

Autoregressors struggle with planning. Suffer from low sampling speeds.

On synthetic tasks[1, 13]



On real-world tasks

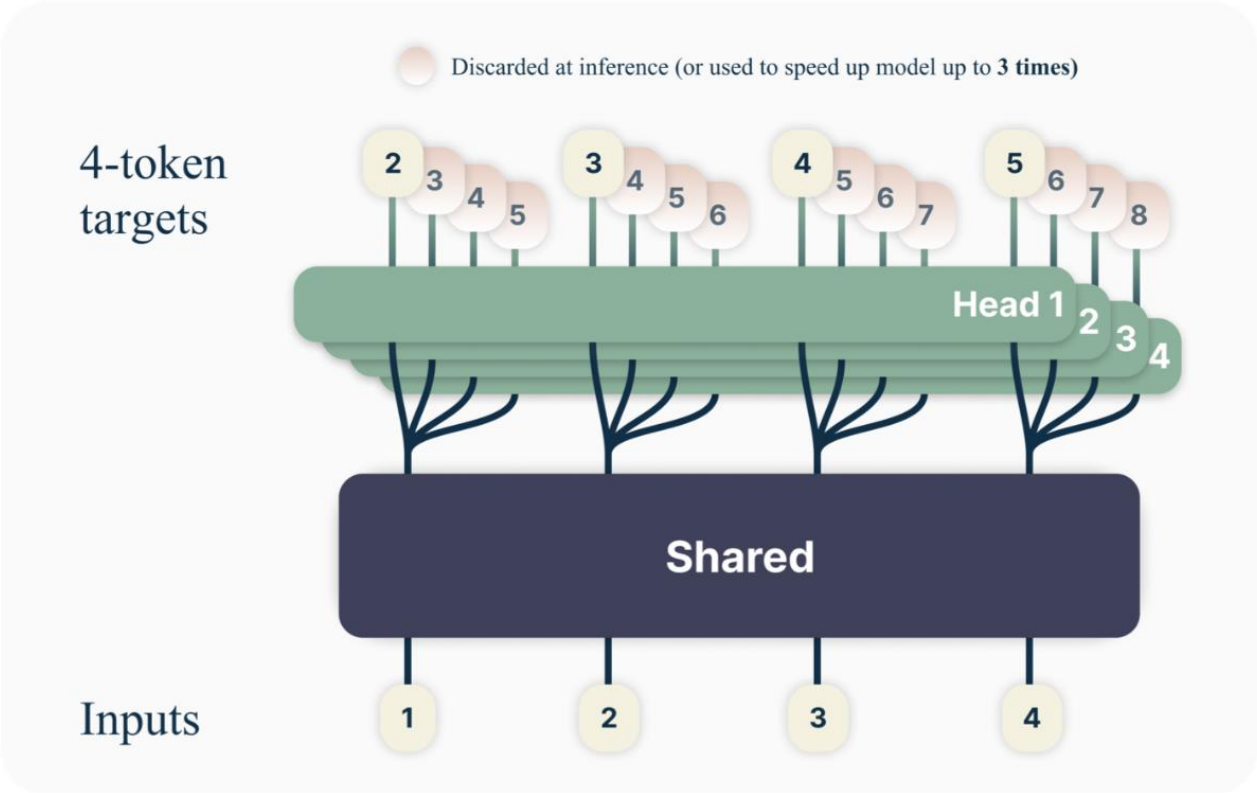
```python
import math

# ...

values = np.mean(values) # Missing import
```

Improvements

- **Speculative Decoding**[7]: N-gram based matching.
- **Medusa**[2]: Frozen-model adaptation for MTP.
- **Multi-Token Prediction** [4]: Introduced by Meta, used in DeepSeek training.



GOSIM

GOSIM AI Paris 2025

Tokenization in LLMs

Tokenizer Choice: Negligible or Crucial?

- The tokenizer vocabulary defines the model's view of language/code. It has important ramifications on performance.
- Tokenizers model token-to-token transitions that may improve or hinder model learning.
- Experiments with training LLMs on raw token-stream of the Go language improves performance.

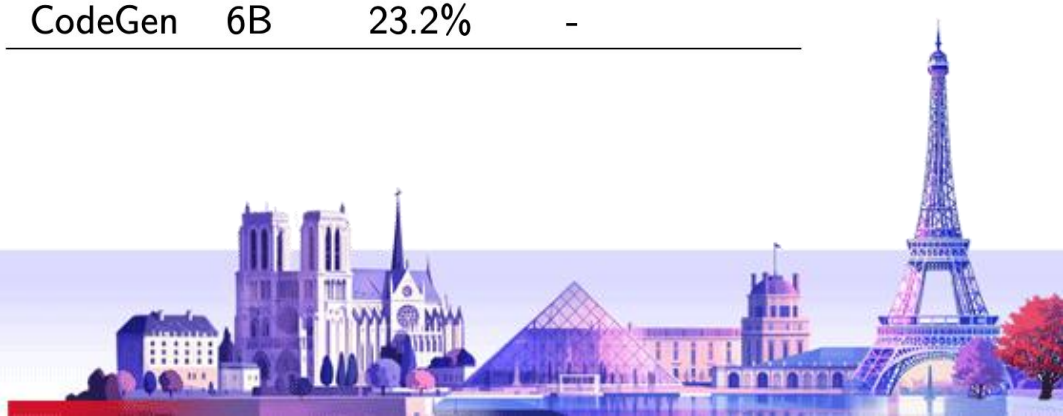
HuggingFace Baseline (BPE)

```
func Tokenize(src string) []string {
    return strings.Split(src, " ")
}
```

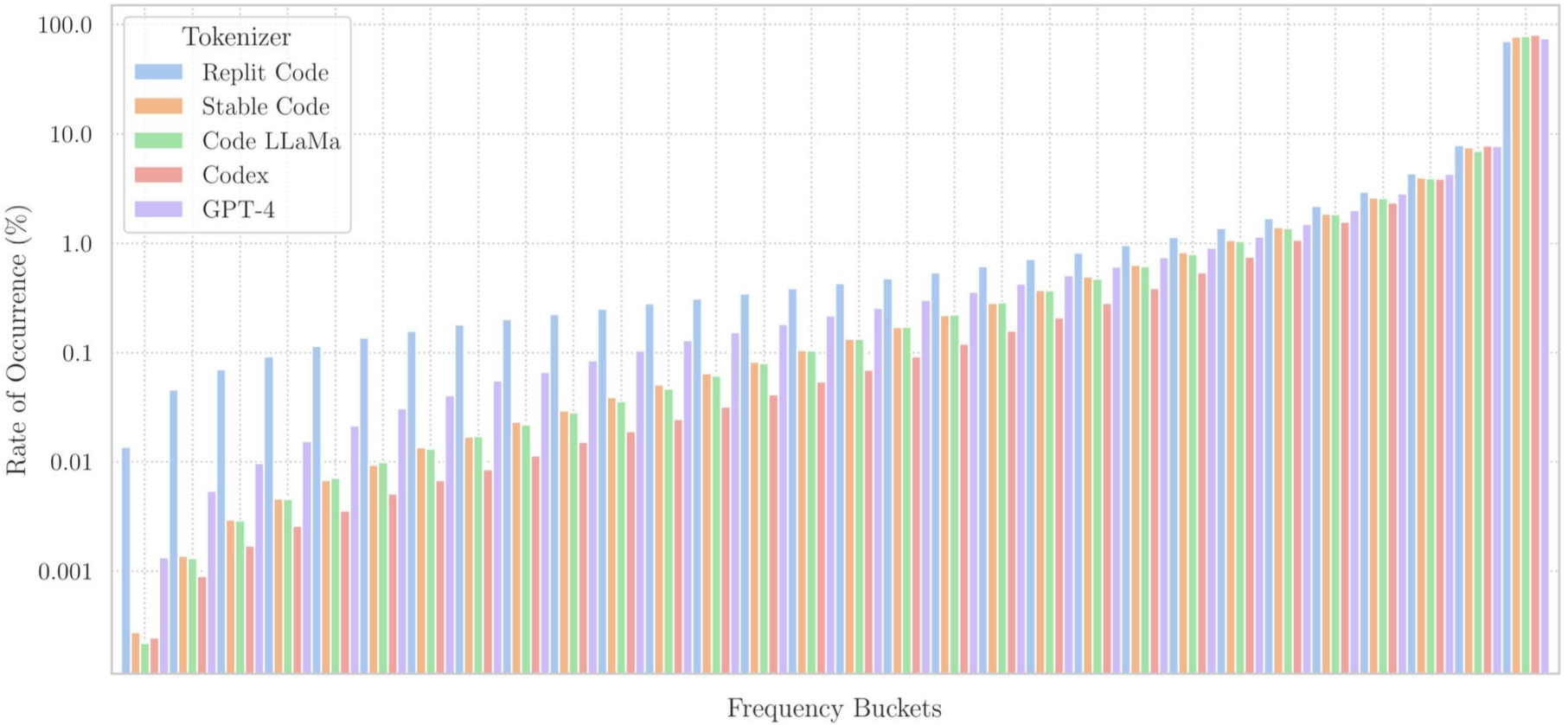
Our Custom Tokenizer

```
func Tokenize(src string) []string {
    return strings.Split(src, " ")
}
```

Model	Size	pass@10	compile@10
Baseline	290M	5.4 %	70.0%
Gopilot	290M	7.3%	77.4%
InCoder	7B	13.8%	-
CodeGen	6B	23.2%	-



Tokenizers are Inefficient



LLM vocabularies now up to **256,000 entries**!

Size of PaLM's per-batch LM head activations (4M toks, BF16)

$$1.024 \times 10^{12} \times 2 = 2.048 \times 10^{12} \text{ bytes}$$

$$= 2.048 \text{ TB}$$

Tokenizers are Clunky

GOSIM

Tokens inflexibly define the **granularity** of decoding and other mechanisms such as attention. Fixed FLOPS per token.

Tokenization is to blame for inability of current models to perform **string manipulation**, or answer questions about **token contents**.

Despite inefficiences, large vocabularies (up to 12.8 million) seem to improve performance[5].

Why Go Beyond Next Token Prediction? GOSIM

The speed wall

3.5TB/s
H100 SXM Mem. BW

590:1
FP8 FLOP:byte ratio

Each forward pass loads
KV Cache + Model Weights

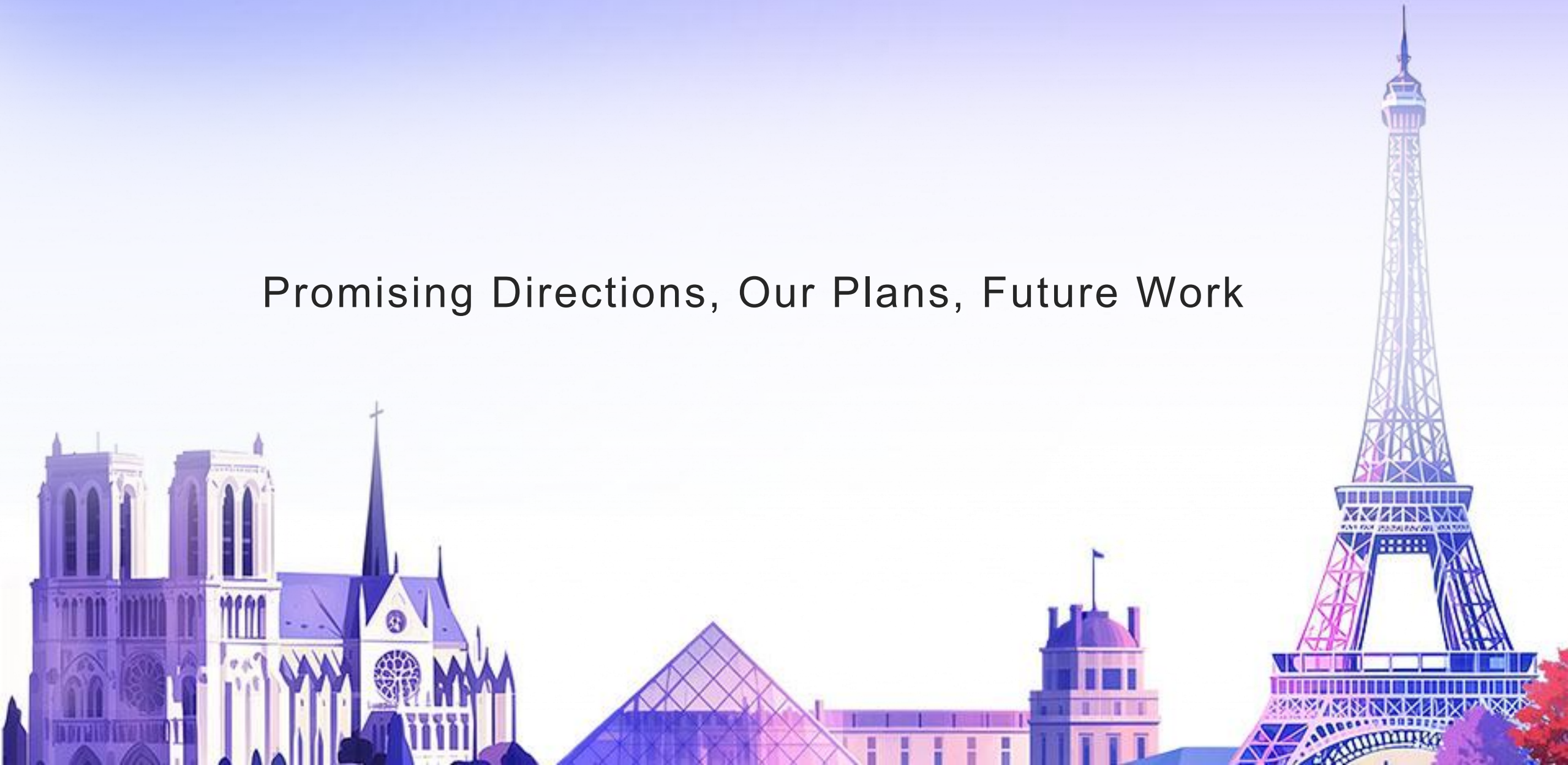
Searching for an architecture that can address this issue

Predict **whole sequences** in parallel instead of **tokens**, one by one.

Diffusion Language Models

Tokenizer-Free Architectures

Promising Directions, Our Plans, Future Work



Diffusion

Diffusion is composed of a **forward noising process** and a **reverse denoising process**.

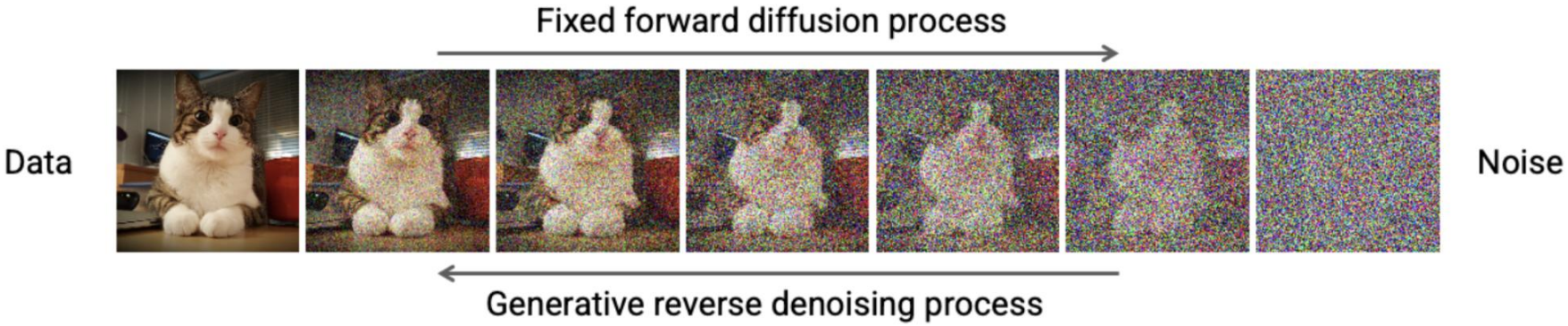


Figure: Continuous diffusion applied to images

$$x_t = x_{t-1} + \sigma_t \epsilon$$

Diffusion Language Models

GOSIM

Applied to text the noising process is often implemented with **discrete absorbing states** (masking) or **random perturbations** to x.

Dif	[MASK]	_language	[MASK]

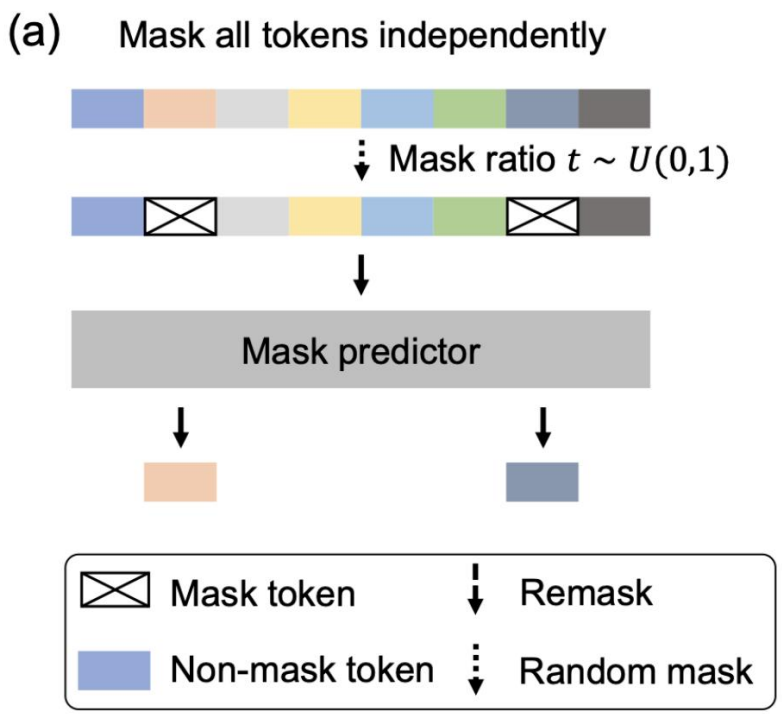
Figure: Mask token noising

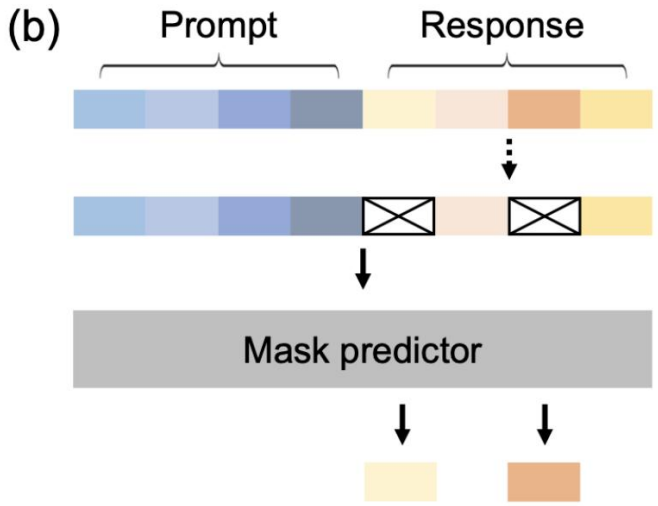
Dif	GoldenMagikarp	_language	!

Figure: Random token noising

Diffusion Language Models: Training

An example with LLaDA[10]. Tokens are randomly masked with probability p. Loss is computed on masked tokens.





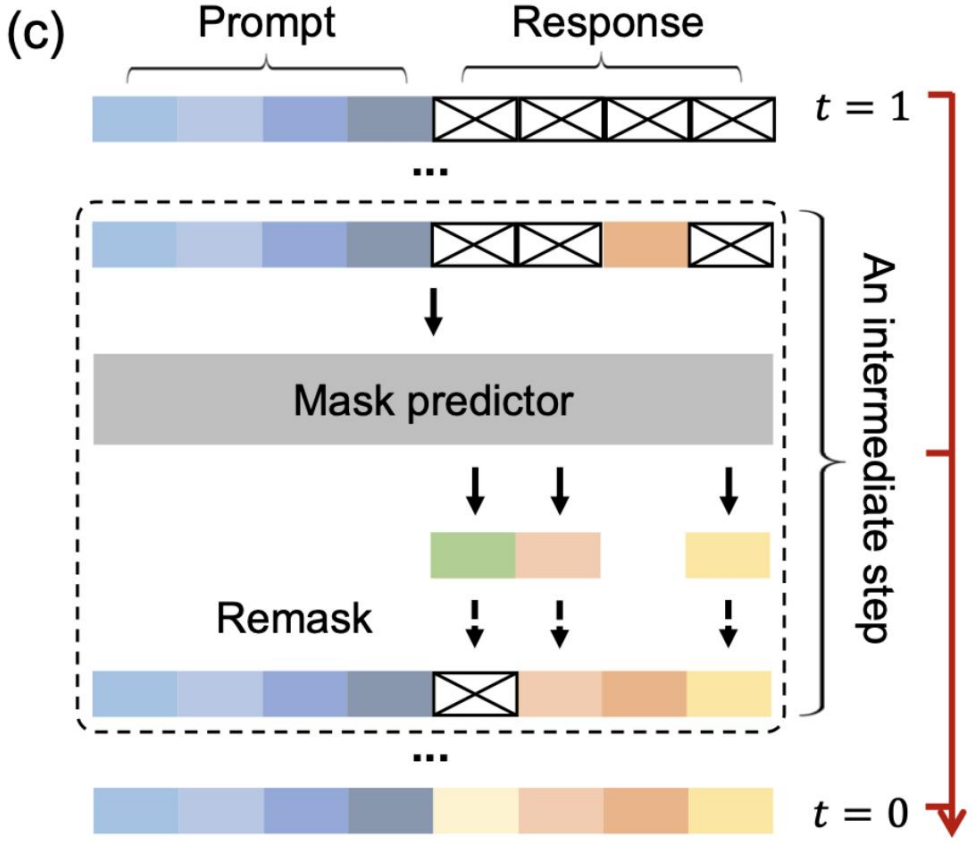
Diffusion Language Models: Inference

GOSIM

At each step, make a prediction for all masked tokens.

- **Hyperparameters:** number of denoising steps, schedule.
- **Remasking:** random or low-confidence tokens are remasked.

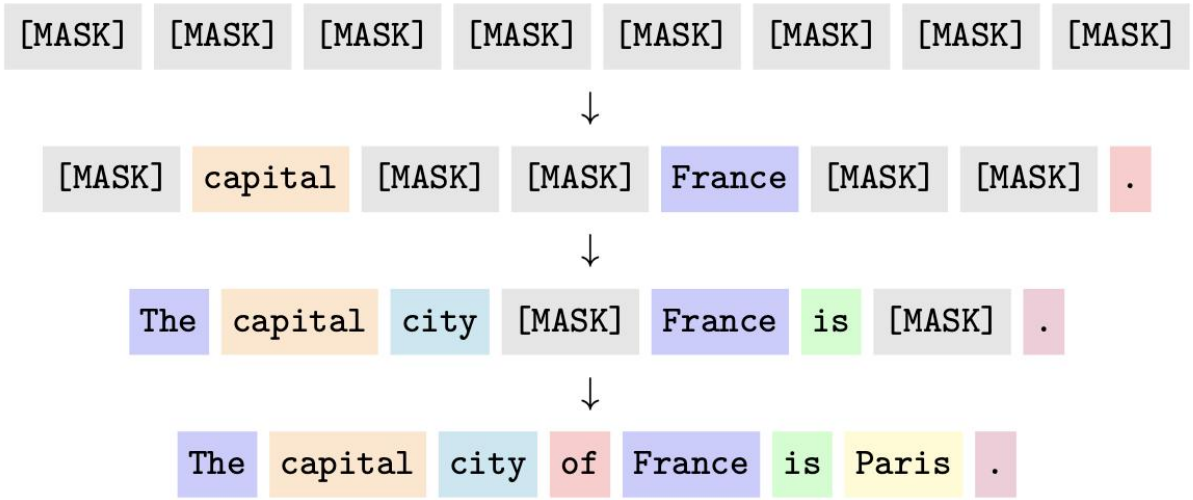
Faster inference when num_steps $\leq$ num_tokens. Can trade speed for quality.



Pitfalls of Discrete Diffusion

In discrete diffusion, noising/denoising is modeled by *"stochastic jumps between states in Markov chains"* which prevents *"iterative refinement"*.

"Discrete diffusion directly jumps from a token to the masked token and vice versa where a wrong jump is non-revokable".



continuous

Continuous Diffusion for Language

Works such as **Diffusion LM** perform the noising process on latent word embeddings rather than on discrete tokens.

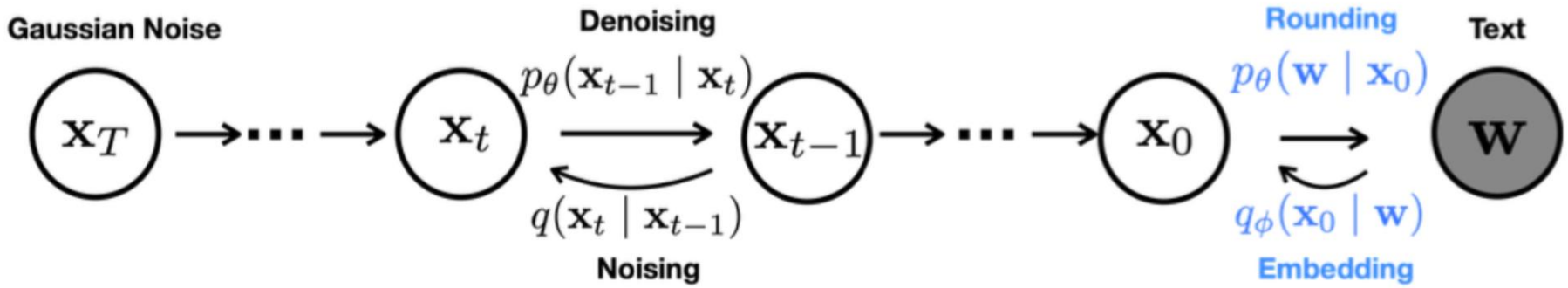


Figure: Example from DiffusionLM[8]

Tokenizer-Free Architectures

"The best tokenizer, is no tokenizer"

Tokenizer-free models don't work with a fixed vocabulary/embeddings. Instead, they operate on "patches", groups of bytes/characters.

Rule-Based

`Hello` `_World`

Fixed-Size

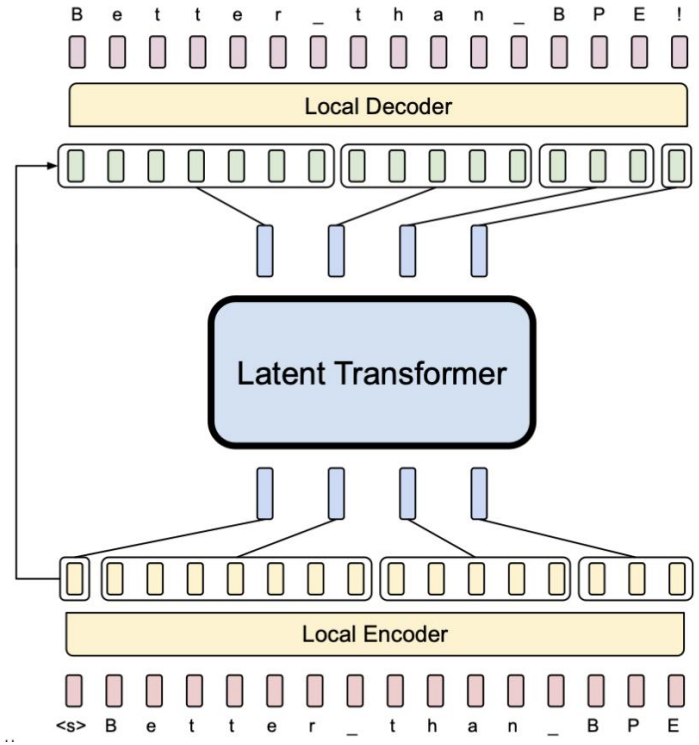
`Hell` `o_Wo` `rld`

Entropy-Based

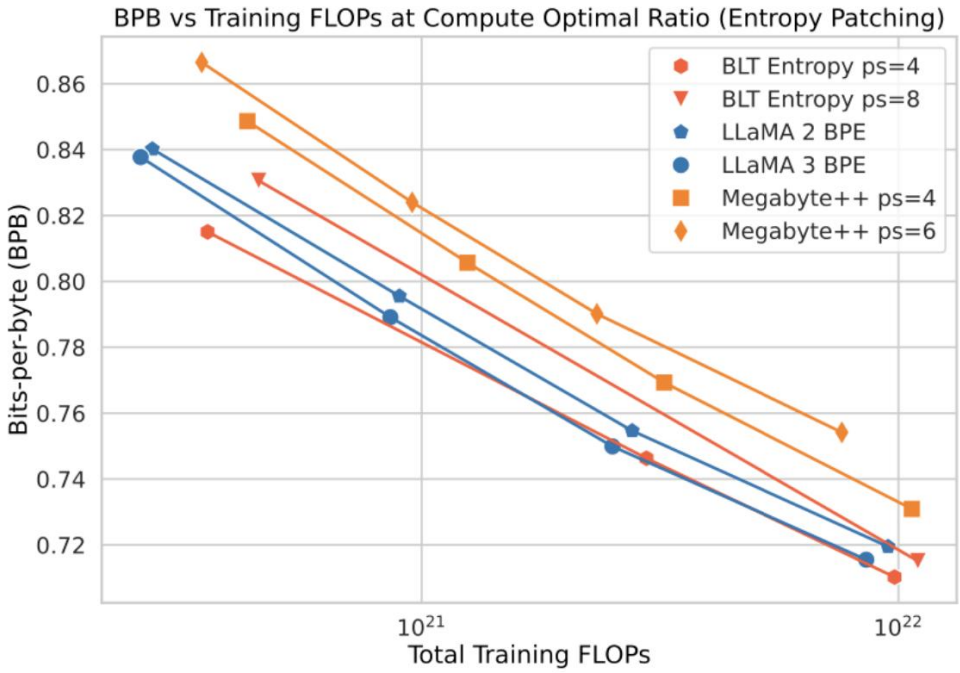
`H` `el` `lo_World`

Byte-Latent Transformer

BLT[11] was introduced by Meta in 2024, based on prior work in tokenizer-free LM (Megabyte[14], Hourglass Transformer[9]).

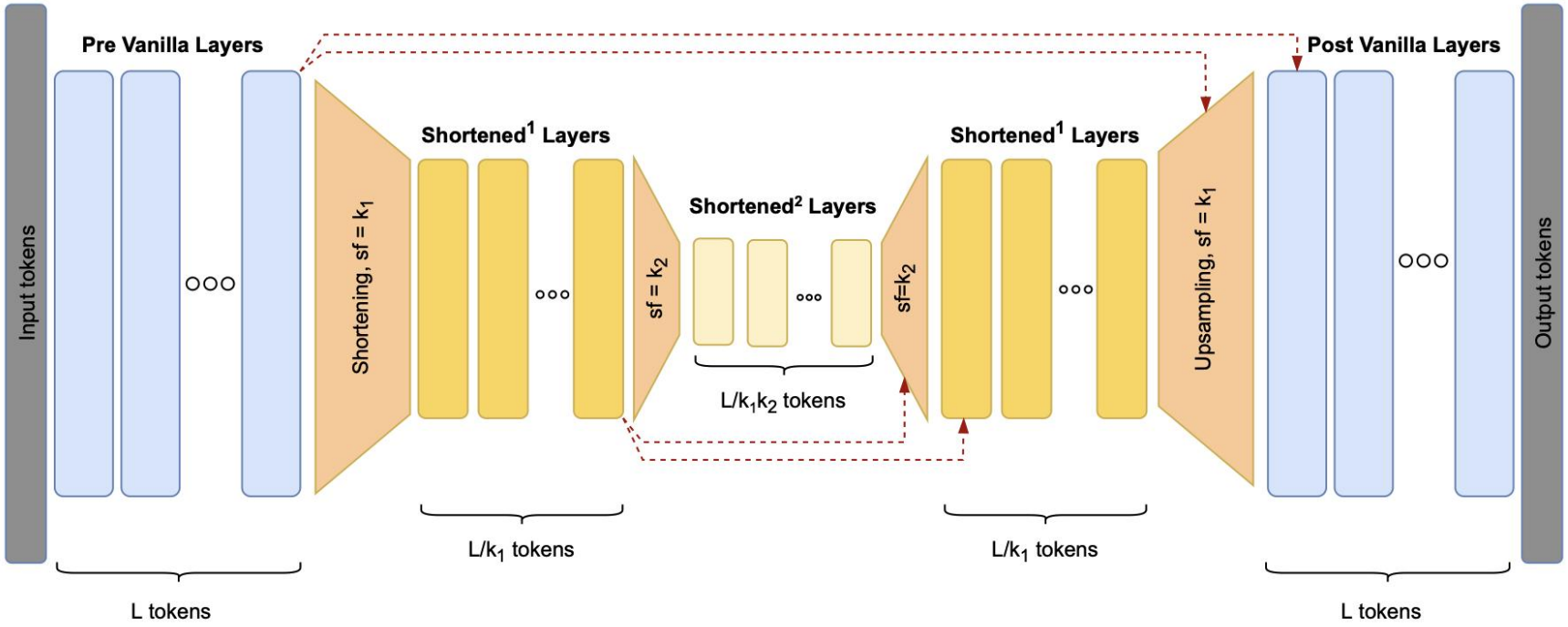


1.6x-1.7x faster with large patches



Hierarchical Transformers

Tokenizer-free models pool patches of bytes to form higher-level word representations.
But why stop here?



Our Plans & Future Work

Scaling Laws | Convergence | Training/Inference Speed | Speed/Quality Tradeoff | Objectives

- Comprehensive ablation study of diffusion training objectives, hyperparameters, architectures
- Integration with pre-trained tokenizer-free models such as Meta's Byte-Latent Transformer[11]
- Technical report on DDLM/CDLM in June

Z.ai & THUKEG on HuggingFace

Conclusion

- **NTP is too slow:**
  - Predicts one token at a time instead of in parallel.
  - Requires moving model parameters, KV-cache to/from HBM for every token.
  - Complexity of attention determined by sequence length, number of tokens.
- **Diffusion:**
  - Decodes N tokens in parallel with $N >> \texttt{steps}$.
  - Better planning ability.
  - No fixed FLOPs per token.
- **Tokenizer-Free:**
  - Flexible language modeling.
  - Trade performance for speed with increasing patch size.
  - Building hierarchical modules, reduce attention complexity.
- **Focus on Code:**
  - Optimized for specific use-case.
  - No silver bullets.

GOSIM

GOSIM AI Paris 2025

Announcement

Company Announcement

If you're interested, write to **diego@labrouste.net**.

Feedback

GOSIM

rojasdiego.com/gosim

Please consider answering this short (>1m) feedback form to help me improve.

GOSIM

GOSIM AI Paris 2025

THANK YOU

References

Gregor Bachmann and Vaishnavh Nagarajan.
The pitfalls of next-token prediction, 2024.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao.
Medusa: Simple llm inference acceleration framework with multiple decoding heads, 2024.

Pavel Chizhov, Catherine Arnett, Elizaveta Korotkova, and Ivan P. Yamshchikov.
Bpe gets picky: Efficient vocabulary refinement during tokenizer training, 2024.

Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Rozière, David Lopez-Paz, and Gabriel Synnaeve.
Better & faster large language models via multi-token prediction, 2024.

Hongzhi Huang, Defa Zhu, Banggu Wu, Yutao Zeng, Ya Wang, Qiyang Min, and Xun Zhou.
Over-tokenized transformer: Vocabulary is generally worth scaling, 2025.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li.
Large language diffusion models, 2025.

Artidoro Pagnoni, Ram Pasunuru, Pedro Rodriguez, John Nguyen, Benjamin Muller, Margaret Li, Chunting Zhou, Lili Yu, Jason Weston, Luke Zettlemoyer, Gargi Ghosh, Mike Lewis, Ari Holtzman, and Srinivasan Iyer.
Byte latent transformer: Patches scale better than tokens, 2024.

Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever.
Consistency models, 2023.

Jiacheng Ye, Jiahui Gao, Shansan Gong, Lin Zheng, Xin Jiang, Zhenguo Li, and Lingpeng Kong.
Beyond autoregression: Discrete diffusion for complex reasoning and planning, 2025.

Taku Kudo.
Subword regularization: Improving neural network translation models with multiple subword candidates, 2018.

Yaniv Leviathan, Matan Kalman, and Yossi Matias.
Fast inference from transformers via speculative decoding, 2023.

Xiang Lisa Li, John Thickstun, Ishaan Gulrajani, Percy Liang, and Tatsunori B. Hashimoto.
Diffusion-lm improves controllable text generation, 2022.

Piotr Nawrot, Jan Chorowski, Adrian Lancucki, and Edoardo Maria Ponti.
Efficient transformers with dynamic token pooling.
In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, page 6403–6417. Association for Computational Linguistics, 2023.

Lili Yu, Dániel Simig, Colin Flaherty, Armen Aghajanyan, Luke Zettlemoyer, and Mike Lewis.
Megabyte: Predicting million-byte sequences with multiscale transformers, 2023.