Meeting 6: NanoGPT — Membangun LLM dari Nol

AI-40X: Generative AI & Large Language Models

Hendri Karisma, M.T.
Dosen Teknik Informatika STMIK Tazkia
VP Engineering at jejakin.com, 2026

Semester 4 — 2025/2026

Outline

Apa itu NanoGPT?

Repositori edukasi oleh Andrej Karpathy

- "The simplest, fastest repository for training/finetuning medium-sized GPTs."
- **Filosofi:** Simple, hackable, educational.
- Hanya ≈300 baris kode untuk model GPT lengkap!
- Bisa mereproduksi GPT-2 (124M parameter).
- Video legendaris: "Let's build GPT: from scratch, in code, spelled out" (2+ jam, 10M+ views).

images/nanogpt_struct.png

Struktur File NanoGPT

File	Fungsi
`model.py`	Definisi arsitektur GPT (CausalSelfAttention, MLP, Block, GPT)
`train.py`	Training loop: data loading, optimizer, checkpointing, logging
`sample.py`	Generate teks dari model yang sudah dilatih
`config/`	File konfigurasi hyperparameter untuk berbagai ukuran model
`data/`	Script persiapan dataset (Shakespeare, OpenWebText, dll.)

Kenapa NanoGPT?

Model production seperti Llama memiliki puluhan ribu baris kode dengan optimasi yang mengaburkan logika inti. NanoGPT menghapus semua "noise" dan menunjukkan esensi murni dari GPT.

Komponen 1: CausalSelfAttention

Inti dari decoder-only transformer — mencegah model "mengintip" masa depan.

CausalSelfAttention (Pseudocode)

```
class CausalSelfAttention(nn.Module):
    def __init__(self, config):
        self.c_attn = nn.Linear(n_embd, 3 * n_embd)  # Q, K, V projection
        self.c_proj = nn.Linear(n_embd, n_embd)       # Output projection
        # Causal mask: segitiga bawah
        self.register_buffer("bias",
            torch.tril(torch.ones(block_size, block_size)))

    def forward(self, x):
        q, k, v = self.c_attn(x).split(n_embd, dim=2)
        # Reshape for multi-head: (B, T, C) -> (B, nh, T, hs)
        att = (q @ k.transpose(-2, -1)) * (1.0 / math.sqrt(k.size(-1)))
        att = att.masked_fill(self.bias[:T, :T] == 0, float('-inf'))
        att = F.softmax(att, dim=-1)
        y = att @ v  # (B, nh, T, hs)
        return self.c_proj(y)  # Combine heads & project
```

Flash Attention: Jika PyTorch $\geq$ 2.0, gunakan `F.scaled_dot_product_attention()` — 2–4$\times$ lebih cepat, hemat memori (tidak menyimpan attention matrix penuh).

Komponen 2: MLP (Feed-Forward Network)

Memproses output attention secara non-linear. Memegang ≈2/3 total parameter model!

MLP (Pseudocode)

```
class MLP(nn.Module):
    def __init__(self, config):
        self.c_fc   = nn.Linear(n_embd, 4 * n_embd)   # Expand 4x
        self.gelu   = nn.GELU()                        # Non-linear activation
        self.c_proj = nn.Linear(4 * n_embd, n_embd)    # Project back

    def forward(self, x):
        x = self.c_fc(x)       # (B, T, n_embd) -> (B, T, 4*n_embd)
        x = self.gelu(x)       # Non-linearity
        x = self.c_proj(x)     # (B, T, 4*n_embd) -> (B, T, n_embd)
        return x
```

- **Kenapa 4×?** Proyeksi ke dimensi lebih tinggi memungkinkan transformasi non-linear yang lebih ekspresif, lalu dikompres kembali.
- **GELU vs ReLU:** GELU (Gaussian Error Linear Unit) lebih smooth — standar di semua LLM modern.

Komponen 3: Block (Satu Layer Transformer)

"Pre-Norm" Transformer Block — urutan yang digunakan GPT-2 dan semua LLM modern.

Block (Pseudocode)

```
class Block(nn.Module):
    def __init__(self, config):
        self.ln_1 = nn.LayerNorm(n_embd)
        self.attn  = CausalSelfAttention(config)
        self.ln_2 = nn.LayerNorm(n_embd)
        self.mlp   = MLP(config)

    def forward(self, x):
        x = x + self.attn(self.ln_1(x))    # Residual + Attention
        x = x + self.mlp(self.ln_2(x))     # Residual + MLP
        return x
```

- **Pre-Norm:** LayerNorm *sebelum* sub-layer (bukan sesudah seperti paper asli). Lebih stabil untuk training model besar.
- **Residual Connection:** `x = x + sublayer(x)` — memungkinkan gradient mengalir langsung, mencegah vanishing gradient.

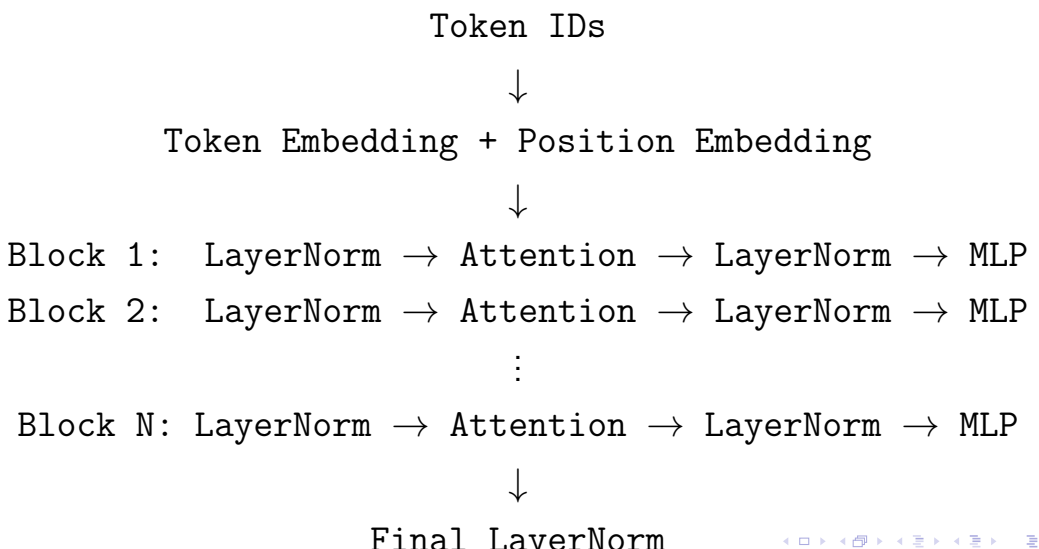
Komponen 4: GPT — Model Lengkap

GPT (Pseudocode)

```
class GPT(nn.Module):
    def __init__(self, config):
        self.transformer = nn.ModuleDict(dict(
            wte  = nn.Embedding(vocab_size, n_embd),      # Token Embedding
            wpe  = nn.Embedding(block_size, n_embd),      # Position Embedding
            h    = nn.ModuleList([Block(config) for _ in range(n_layer)]),
            ln_f = nn.LayerNorm(n_embd),                  # Final LayerNorm
        ))
        self.lm_head = nn.Linear(n_embd, vocab_size, bias=False)

    def forward(self, idx, targets=None):
        tok_emb = self.transformer.wte(idx)               # (B, T, C)
        pos_emb = self.transformer.wpe(positions)         # (T, C)
        x = tok_emb + pos_emb
        for block in self.transformer.h:
            x = block(x)                                  # N layers
        x = self.transformer.ln_f(x)
        logits = self.lm_head(x)                          # (B, T, vocab_size)
        loss = F.cross_entropy(logits.view(-1, logits.size(-1)),
                               targets.view(-1)) if targets else None
        return logits, loss
```

Diagram Alur: Dari Token ke Prediksi



AdamW Optimizer & Weight Decay

Kenapa AdamW, bukan Adam biasa?

- **Adam** (Kingma & Ba, 2014): adaptive learning rate per parameter — standar deep learning.
- **Masalah:** Implementasi weight decay di Adam (L2 regularization) *salah* — weight decay tercampur dengan adaptive gradient.
- **AdamW** (Loshchilov & Hutter, 2019): *decoupled* weight decay — diterapkan langsung ke bobot, bukan ke gradient.

Update Rule AdamW

$$\theta_{t+1} = \theta_t - \eta \left(\frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} + \lambda \theta_t \right)$$

di mana λ adalah weight decay yang *decoupled* dari gradient.

Typical hyperparameters: $\eta = 6 \times 10^{-4}$, $\beta_1 = 0.9$, $\beta_2 = 0.95$, $\lambda = 0.1$

Learning Rate Schedule: Cosine Decay + Linear Warmup

Jangan gunakan learning rate konstan!

1. **Linear Warmup** (awal training):
   - LR naik dari 0 ke $\eta_{\max}$ selama T_w step.
   - Mencegah instabilitas saat bobot masih random.
   - Typical: $T_w = 2000$ steps.
2. **Cosine Decay** (setelah warmup):
   - LR turun perlahan mengikuti kurva cosinus.
   - $\eta_t = \eta_{\min} + \frac{1}{2}(\eta_{\max} - \eta_{\min})(1 + \cos(\pi \cdot t/T))$
   - Typical: $\eta_{\min} = 0.1 \times \eta_{\max}$

Bentuk Kurva LR:

```
|-warmup-->| <- cosine decay ->|
  |/\              \------------------|
```

Naik cepat di awal, turun perlahan hingga akhir.

Gradient Clipping & Mixed Precision

Gradient Clipping

- Cegah **exploding gradients** yang menyebabkan training diverge.
- Teknik: Clip gradient norm ke maksimum tertentu.
- `torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)`
- Jika $\|\nabla\| > 1.0$, gradient di-scale down secara proporsional.

Mixed Precision (FP16/BF16)

- **FP32:** 32-bit floating point (default).
- **BF16:** 16-bit Brain Float — range sama dengan FP32, presisi lebih rendah.
- **Keuntungan:**
  - Memory $\approx$50% lebih hemat.
  - Speed $\approx 2\times$ lebih cepat (Tensor Cores).
- `torch.autocast('cuda', dtype=torch.bfloat16)`
- **BF16 lebih aman** dari FP16 — tidak perlu loss scaling.

Weight Initialization

Inisialisasi bobot yang tepat = fondasi training yang stabil.

- **Default NanoGPT:** Normal distribution $\mathcal{N}(0, 0.02)$ untuk semua linear layers.
- **Residual scaling:** Output projection di attention dan MLP di-scale oleh $\frac{1}{\sqrt{2N}}$, di mana N = jumlah layer.
- **Alasan:** Setiap residual connection *menambahkan* ke input. Tanpa scaling, variance tumbuh $\propto N$ — menyebabkan instabilitas.

Kode Inisialisasi

```
nn.init.normal_(module.weight, mean=0.0, std=0.02)
nn.init.zeros_(module.bias)
```

Untuk residual projection: `std = 0.02 / math.sqrt(2 * n_layer)`

Persiapan Dataset

Kita akan melatih GPT pada dua pilihan dataset:

1. **Shakespeare (Default)**
   - ≈1MB teks drama Shakespeare.
   - Vocab: 65 karakter unik.
   - Cocok untuk demo cepat.

2. **Pantun Indonesia (Tantangan!)**
   - Kumpulkan pantun dari berbagai sumber.
   - Struktur: a-b-a-b (4 baris).
   - Model harus belajar rima dan irama.

Prepare Shakespeare

```
# data/shakespeare_char/prepare.py
with open('input.txt', 'r') as f:
    data = f.read()
chars = sorted(list(set(data)))
stoi = {ch:i for i,ch in enumerate(chars)}
itos = {i:ch for i,ch in enumerate(chars)}
```

Contoh Data Pantun

```
Kalau ada sumur di ladang
Boleh kita menumpang mandi
Kalau ada umur yang panjang
Boleh kita berjumpa lagi
---
Pisang emas dibawa berlayar
...
```

Character-Level Tokenization

Untuk kesederhanaan, kita gunakan tokenisasi level karakter.

Character-Level Encoding/Decoding

```
# Encode: string -> list of integers
def encode(s):
    return [stoi[c] for c in s]

# Decode: list of integers -> string
def decode(l):
    return ''.join([itos[i] for i in l])

# Contoh
text = "Halo dunia"
encoded = encode(text)   # [17, 39, 50, 53, 1, 42, 57, 52, 45, 39]
decoded = decode(encoded) # "Halo dunia"

# Konversi ke tensor PyTorch
import torch
data = torch.tensor(encode(text), dtype=torch.long)
```

- **Kelebihan:** Sederhana, tidak perlu library tokenizer, vocab kecil (≈65–100).
- **Kekurangan:** Sequence panjang, model harus belajar spelling sendiri. Untuk production, gunakan BPE (sudah dipelajari di Meeting 4).

Training di Google Colab

Langkah-langkah:

1. **Clone repository:**

```
!git clone https://github.com/karpathy/nanoGPT.git
%cd nanoGPT
```

2. **Siapkan dataset:**

```
!python data/shakespeare_char/prepare.py
```

3. **Mulai training** (konfigurasi untuk Colab free tier):

```
!python train.py config/train_shakespeare_char.py \
    --device=cuda --compile=True --eval_iters=20 \
    --log_interval=10 --block_size=256 --batch_size=12 \
    --n_layer=4 --n_head=4 --n_embd=128 \
    --max_iters=5000 --lr_decay_iters=5000 \
    --dropout=0.0 --learning_rate=1e-3
```

4. **Generate teks:**

```
!python sample.py --out_dir=out-shakespeare-char \
    --start="To be or not to be"
```

Monitoring: Loss Curves & Generated Samples

Bagaimana tahu training berjalan baik?

1. Loss Curves

- **Train loss** harus turun secara konsisten.
- **Val loss** harus mengikuti train loss.
- Jika val loss naik sedangkan train loss turun $\rightarrow$ **overfitting**!
- Target: train loss $\approx$ 1.0–1.5 untuk Shakespeare char-level.

2. Generated Text Samples

- **Iter 0:** "xkq&#jPz..." (random noise)
- **Iter 500:** "the the the and..." (belajar kata umum)
- **Iter 2000:** "KING: My lord, I..." (struktur dialog)
- **Iter 5000:** Teks koheren dengan format drama Shakespeare!

Tips

Generate sample setiap 500–1000 iterasi untuk melihat perkembangan kualitatif model. Jangan hanya bergantung pada angka loss!

Persiapan UTS — Pilihan Tugas

Pilih salah satu:

Opsi A: NanoGPT dari Nol

- Latih NanoGPT pada dataset pilihan (Shakespeare, Pantun, atau teks Indonesia lain).
- Eksperimen dengan hyperparameter.
- Dokumentasikan loss curve dan sample output.
- **Bonus:** Buat dataset custom (lagu daerah, resep masakan, dll.)

Opsi B: Fine-tuning Model Pre-trained

- Fine-tune model kecil (SmolLM, Qwen2.5-0.5B, dll.) pada task spesifik.
- Gunakan Hugging Face `transformers` + `peft`.
- Dokumentasikan performa sebelum dan sesudah fine-tuning.
- **Bonus:** Gunakan dataset berbahasa Indonesia.

Kriteria Penilaian

- **Pemahaman konsep** (40%): Penjelasan arsitektur, training process, dan keputusan desain.
- **Implementasi** (30%): Kode berjalan, loss turun, output koheren.

Kesimpulan

1. **NanoGPT** membuktikan bahwa GPT bisa dibangun dengan ≈300 baris kode Python — arsitektur LLM *tidak* misterius.
2. **4 komponen inti:** CausalSelfAttention, MLP, Block, dan GPT class — masing-masing punya peran jelas.
3. **Training best practices** (AdamW, cosine LR, gradient clipping, mixed precision) sama pentingnya dengan arsitektur.
4. **Hands-on experience** melatih model sendiri memberikan intuisi yang tidak bisa didapat dari membaca paper saja.
5. **Persiapan UTS:** Pilih Opsi A (NanoGPT) atau Opsi B (Fine-tuning), mulai eksperimen dari sekarang!

Homework

Clone NanoGPT, jalankan training Shakespeare di Google Colab, dan kirimkan screenshot loss curve + generated text ke LMS sebelum Meeting 7.

Referensi

- Karpathy, A. (2023). `nanoGPT` Repository. `https://github.com/karpathy/nanoGPT`
- Karpathy, A. (2023). "Let's build GPT: from scratch, in code, spelled out." YouTube.
- Radford, A., et al. (2019). "Language Models are Unsupervised Multitask Learners." (GPT-2). OpenAI.
- Loshchilov, I. & Hutter, F. (2019). "Decoupled Weight Decay Regularization." (AdamW). *ICLR*.
- Dao, T., et al. (2022). "FlashAttention: Fast and Memory-Efficient Exact Attention." *NeurIPS*.
- Kingma, D. & Ba, J. (2015). "Adam: A Method for Stochastic Optimization." *ICLR*.