LLM-Based Summarization

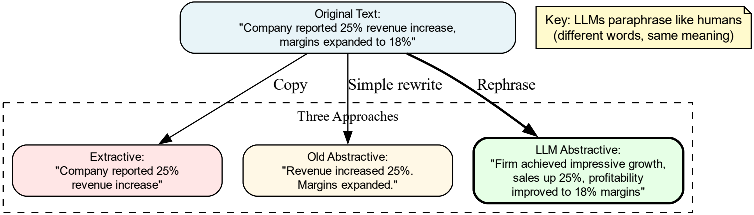
From Paraphrasing to Production

NLP Course 2025

October 31, 2025

Professional Template — Graphviz Flows + Clean Visualizations

The Paraphrasing Challenge



Discovery: LLMs don't just copy sentences - they rephrase like humans

Unlike extractive methods, LLMs generate natural variations of text

What Makes LLMs Different?

Traditional Approaches

Extractive summarization:

- Select important sentences
- Copy verbatim from source
- No generation capability
- Limited coherence

Old neural models (BART, T5):

- Trained for specific task
- Fixed behavior
- Limited control

LLM Approach

Instruction-following models:

- **Generate** new text
- Natural paraphrasing
- Creative rewording
- Coherent narratives

GPT-3.5/4, Claude, LLaMA:

- Follow natural language instructions
- Highly controllable (prompts)
- Flexible (zero-shot/few-shot)

LLMs enable summarization through conversational instructions

Summarization Task Definition

What is Summarization?
Input: Long document (article, report, paper)
Output: Concise text capturing key information

Requirements:

- Preserve main ideas
- Remove redundancy
- Maintain coherence
- Target length (e.g., 3 sentences, 150 words)

LLM Advantages

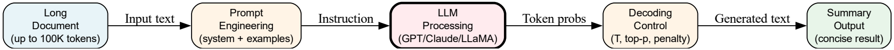
1. Natural language control
"Summarize in 3 sentences"
"Focus on policy implications"
"Write for general audience"

2. Adaptable style
Formal, casual, technical, simple

3. Context-aware
Can combine multiple documents
Handle different domains

LLMs make summarization accessible through simple prompts

The Summarization Pipeline



Three Control Points: Prompt design, model selection, decoding parameters

Each stage offers levers for controlling summary quality and style

Zero-Shot Prompting: The Simplest Approach

Concept: Give direct instructions without examples

Example:

```
Prompt: "Summarize the following article in 3 sentences:

[800-word article about Federal Reserve rates]

Focus on main findings and policy implications."
```

Output: "Federal Reserve chiefs have raised interest rates to a range of 5.00% to 5.25%, the highest level in 16 years."

Key Insight: Just ask! No training examples needed

Zero-shot works when task is clear and model has seen similar examples during pre-training

Few-Shot Prompting: Teaching by Example

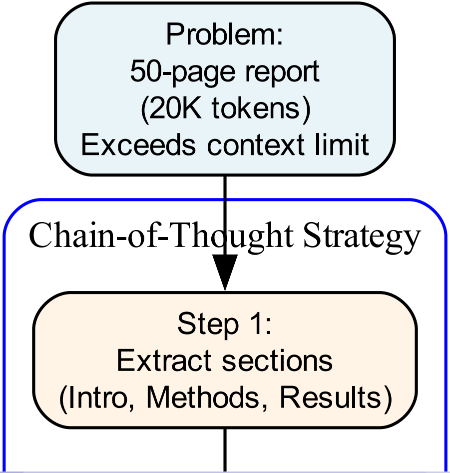
Concept: Provide 2-5 examples to teach format and style

```
Prompt: "You are a financial news summarizer.

Example:
Article: Stock market rose 2% on tech earnings...
Summary: Markets gained on tech earnings. Indexes up 2%.

Now summarize this article in the same style:"
```

Output: "Federal Reserve raises interest rates by 0.25 percentage points on Wednesday, a pause that continues to be a trend between the central bank and the central bank."

Key Insight: Show 2-5 examples → model learns your style

Few-shot dramatically improves quality when you need specific format or tone


Problem:
50-page report
(20K tokens)
Exceeds context limit
Chain-of-Thought Strategy
Step 1:
Extract sections
(Intro, Methods, Results)

Prompt Engineering Best Practices

Structure Your Prompt

1. System role (optional)
"You are a professional technical writer"

2. Task instruction
"Summarize the following article"

3. Constraints
"In exactly 3 sentences"
"Focus on main findings"

4. Examples (if few-shot)
Show 2-3 complete examples

5. Input text
Paste document to summarize

Common Patterns

Length control:
"Summarize in X sentences/words"

Focus specification:
"Highlight policy implications"
"Explain for non-experts"

Style guidance:
"Use formal academic tone"
"Write conversationally"

Format requirements:
"Output as bullet points"
"Include a title"

Good prompts are specific, structured, and include desired output format

Worked Example: Prompt Evolution

Task: Summarize research paper on climate change

Attempt 1 (vague):
"Summarize this paper" → Too general, inconsistent quality

Attempt 2 (better):
"Summarize this climate research paper in 3 sentences, focusing on main findings and policy recommendations" → Better, but still varies

Attempt 3 (best):
"You are a science journalist. Summarize this climate research paper in exactly 3 sentences for a general audience. Structure: (1) main finding, (2) evidence, (3) policy implication. Use plain language, no jargon." → **Consistent, high quality**

Iterative refinement: vague → specific → structured with role and format

Quick Self-Check

Question: You need to summarize 100 medical research papers for a literature review. Which approach?

A) Zero-shot with simple prompt

B) Few-shot with 3 examples of your desired format

C) Chain-of-thought for each paper

D) Different prompt for each paper

Answer: **B** - Few-shot with examples

Reasoning:

- Need consistent format across 100 papers
- Medical domain benefits from examples
- Zero-shot varies too much
- CoT unnecessary (papers likely fit in context)
- Consistency ¿ customization for batch processing

Choose prompting strategy based on volume, consistency needs, and domain specificity

Decoding Parameters: Fine-Tuning Output

What is Decoding?
LLM generates text token-by-token:

Step 1: Compute probabilities
$P(w_1|context) = [0.35, 0.25, 0.15, ...]$
Step 2: Select next word
Different strategies $\rightarrow$ different outputs
Step 3: Repeat until done
Stop at max_tokens or natural end

Key point: Same prompt + model, different parameters $\rightarrow$ very different summaries

Main Parameters
1. Temperature (T)
Controls randomness
Low (0.3): safe, repetitive
High (1.0): creative, varied

2. Top-p (nucleus)
Dynamic probability cutoff
Typical: $p = 0.9$

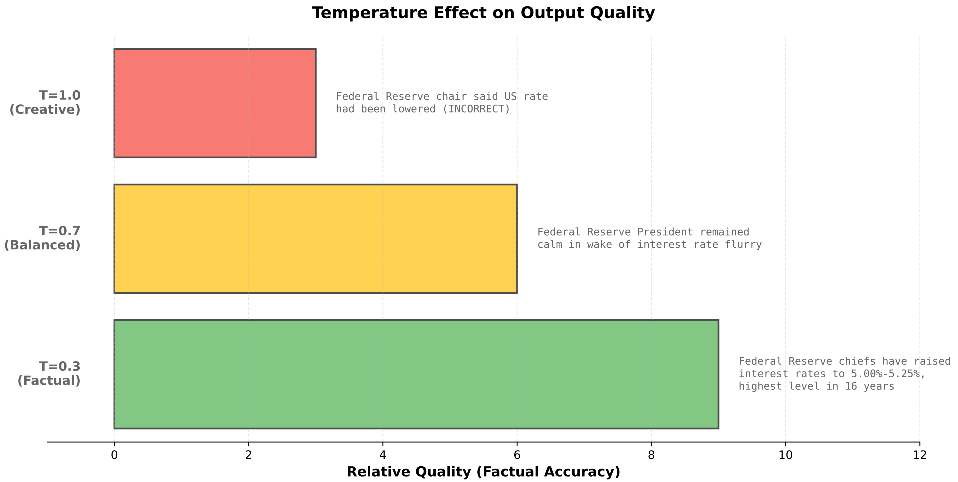
3. Max tokens
Length limit (e.g., 150)

4. Repetition penalty
Reduce redundancy (1.1-1.2)

5. Stop sequences
End generation early

Decoding parameters are your knobs for controlling output quality and diversity

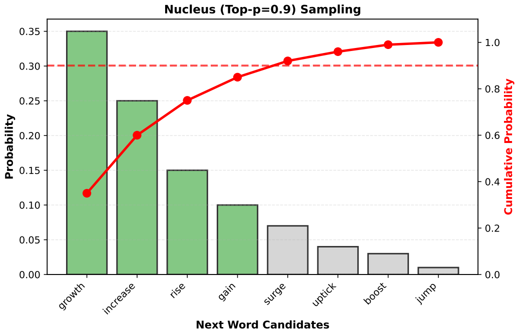
Temperature: Randomness Control



Key Insight: Lower T = factual accuracy — Higher T = creative variation

For summarization: Use T=0.3-0.5 to prioritize accuracy over creativity

Top-p (Nucleus Sampling): Dynamic Cutoff




```
Top-p Algorithm:

1. Sort words by probability
2. Compute cumulative sum
3. Include words until sum ≥ p (0.9)
4. Sample from included set

Result: Dynamic vocabulary size
- Peaked distribution → few words
- Flat distribution → many words

Included (green): Top 90% probability
Excluded (gray): Bottom 10% (too unlikely)
```

Key Insight: Include words until cumulative probability reaches p (e.g., 0.9)

Adapts to probability distribution: peaked → fewer words, flat → more words

Max Tokens: Length Control

Max Tokens: Length Control Comparison

max_tokens=50 (Too Short)

```
The study examined...
[TRUNCATED]
```

X

max_tokens=150 (Just Right)

```
Study examined treatment efficacy in
1000 patients. Results showed 25%
improvement with minimal side effects.
Recommended for clinical use.
```

OK

max_tokens=500 (Too Verbose)

```
The comprehensive longitudinal study
meticulously examined treatment efficacy
across multiple patient cohorts totaling
approximately 1000 individuals.
Results demonstrated statistically
significant improvement of 25%...
```

?

Set max_tokens based on desired summary length (typically 100-200 for articles)

Key Insight: Set based on desired summary length (100-200 typical for news)

Repetition Penalty: Avoiding Redundancy

Repetition Penalty Effect

WITHOUT Penalty (1.0)

`"The company reported strong results. The company announced strong earnings. The company's financial performance was strong. The company showed strong growth."`

5x "company"
5x "strong"

↓ *Apply penalty=1.2*

WITH Penalty (1.2)

`"The firm reported strong Q4 results, with revenue increasing 15% year-over-year. This performance exceeded analyst expectations and demonstrated effective cost management."`

Varied vocabulary
Natural flow

Repetition Penalty: Reduces probability of recently used tokens

Values: 1.0 (none) | 1.1 (mild) | 1.2 (moderate) | 1.5+ (aggressive)

For summarization: Use 1.1-1.2 to encourage diversity without awkwardness

Key Insight: Penalty 1.1-1.2 encourages vocabulary diversity

Worked Example: Parameter Tuning

Scenario: Summarizing financial earnings reports (need accuracy, not creativity)

Configuration 1 (default):
$T = 1.0$, $p = 1.0$, repetition_penalty=1.0
Result: "The company performed well and results were good..."
Too vague, repetitive

Configuration 2 (optimized):
$T = 0.3$, $p = 0.9$, max_tokens=150, repetition_penalty=1.2
Result: "Q4 revenue increased 18% to $2.1B, exceeding analyst expectations of $1.9B. Operating margins expanded from 12% to 15% due to cost optimization. Management raised full-year guidance by 10%."
Specific, concise, accurate

Low temperature + repetition penalty = accurate, non-redundant financial summaries

Decoding Best Practices by Use Case

Use Case	Temp	Top-p	Max Tokens	Rep. Penalty
News articles	0.3-0.5	0.9	100-150	1.1-1.2
Scientific papers	0.3	0.85	200-300	1.2
Legal documents	0.2	0.8	300-500	1.1
Customer reviews	0.5-0.7	0.9	50-100	1.2
Meeting transcripts	0.4	0.9	150-250	1.3
Creative content	0.7-1.0	0.95	variable	1.0-1.1

General Rules:

- **Factual domains** (news, science, legal): Low temp (0.2-0.5), higher repetition penalty
- **Creative domains** (marketing, content): Higher temp (0.7+), lower penalty
- **Length**: Match typical summary length for domain
- **Top-p**: Usually 0.85-0.95 (rarely need to change)

Start conservative (T=0.3, p=0.9), then increase creativity if needed

Long Documents: The Context Window Problem

The Problem

Most LLMs have limited context:

- GPT-3.5: 4K tokens (3K words)
- GPT-4: 8K-32K tokens
- GPT-4 Turbo: 128K tokens
- Claude 2: 100K tokens

Real-world documents:

- PhD thesis: 50K-100K words
- Legal contract: 20K-50K words
- Research report: 10K-30K words

Many documents exceed context limits!

Three Strategies

1. Chunking

Split → Summarize each → Merge
Simple, parallelizable

2. Map-Reduce

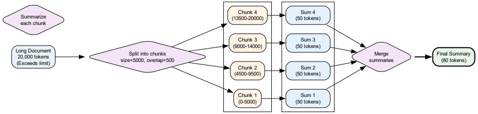
Map: Process all chunks independently
Reduce: Combine into final output
Scalable to many documents

3. Recursive Hierarchical

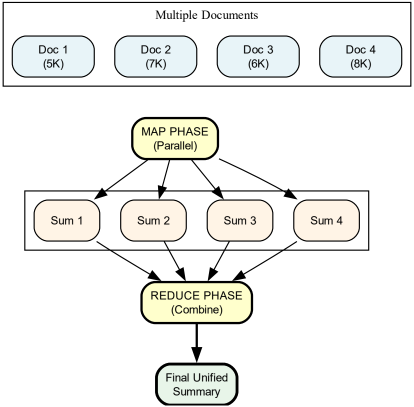
Level 0: Summarize sections
Level 1: Combine summaries
Level 2: Final synthesis
Best coherence, preserves structure

Strategy choice depends on document length, structure, and coherence requirements

Chunking and Map-Reduce Strategies



Chunking: Sequential processing
Good for: Single long document



Map-Reduce: Parallel processing
Good for: Multiple documents

Key Takeaways

1. **LLMs enable human-like paraphrasing** - not just sentence extraction
2. **Prompt engineering is critical** - zero-shot, few-shot, chain-of-thought
3. **Decoding parameters control output** - temperature, top-p, max_tokens, repetition penalty
4. **Different use cases need different settings** - factual (low temp) vs creative (high temp)
5. **Long documents need special strategies** - chunking, map-reduce, hierarchical
6. **Iteration improves quality** - refine prompts and parameters based on outputs

Bottom Line: LLM summarization = Prompts + Decoding + Context handling

Modern summarization is about controlling LLMs through natural language

Technical Appendix

Advanced Prompting — Decoding Mathematics — Context Handling

A1: System Prompts and Role-Playing

System Prompts set global behavior (GPT-4, Claude)

Example System Prompt:
``You are an expert medical researcher with 20 years of experience. Summarize clinical studies with focus on methodology, sample size, statistical significance, and clinical implications. Always mention limitations. Use precise medical terminology but explain complex concepts."

Effects:

- Establishes expertise level
- Sets domain vocabulary
- Defines required elements (methodology, limitations)
- Balances technical accuracy with accessibility

Role-Playing Variations:

- "You are a skeptical peer reviewer" $\rightarrow$ critical analysis
- "You are explaining to a patient" $\rightarrow$ simplified language
- "You are a regulatory auditor" $\rightarrow$ compliance focus

System prompts persist across conversation, user prompts are per-request

A2: Format Control and Structured Output

Challenge: Ensure consistent output structure across many summaries

Technique 1 - Template specification:
"Output format:
Title: [One sentence]
Key Findings: [Bullet list of 3-5 items]
Methodology: [One paragraph]
Implications: [One paragraph]"

Technique 2 - JSON output:
"Return summary as JSON: {"title": "...", "findings": ["...", "..."], "methodology": "...", "implications": "..."}"

Benefits:

- Enables automated post-processing
- Ensures all required sections present
- Facilitates database storage
- Allows programmatic validation

Structured output essential for production systems processing thousands of documents

A3: Multi-Step Chain-of-Thought Decomposition

For very complex documents, break reasoning into explicit steps:

Prompt Pattern:
"Let's summarize this 100-page research report step by step:
Step 1: Identify the main research question and hypothesis
Step 2: Extract methodology details (sample, design, procedures)
Step 3: Summarize key findings with supporting evidence
Step 4: Note limitations and caveats mentioned
Step 5: Extract policy or practical recommendations
Step 6: Synthesize all above into 5-sentence executive summary
Please work through each step explicitly, then provide the final summary."

Why this works:

- Forces systematic coverage of all aspects
- Reduces hallucination (grounded in text)
- Makes reasoning transparent and debuggable
- Better handles complex logical relationships

Multi-step prompts improve accuracy but increase token usage (costs)

A4: Self-Consistency and Multiple Samples

Technique: Generate multiple summaries, then combine or select best

Approach 1 - Majority voting:

1. Generate 5 summaries with $T = 0.7$ (moderate diversity)
2. Extract key facts mentioned in each
3. Final summary includes facts appearing in 3+ versions

Approach 2 - Best-of-N selection:

1. Generate 3 summaries with different prompts
2. Use LLM to evaluate: "Which summary is most accurate and comprehensive?"
3. Return selected summary

Approach 3 - Ensemble merging:

1. Generate 3 summaries from different models (GPT-4, Claude, LLaMA)
2. Prompt: "Combine these 3 summaries into one optimal summary"
3. Leverage strengths of multiple models

Multiple samples reduce single-run errors but increase computational cost

A5: Prompt Optimization and Iteration

Systematic prompt improvement:

Phase 1 - Baseline (5 test documents):
Prompt: "Summarize this article in 3 sentences"
Evaluate: Generic, misses key points 40% of time

Phase 2 - Add specificity:
Prompt: "Summarize focusing on: (1) main finding, (2) evidence, (3) implications"
Evaluate: Better coverage, still inconsistent phrasing

Phase 3 - Add examples and format:
Prompt: "¡System role + 2 examples¿ Summarize this article. Format: Finding: ... — Evidence: ... — Implications: ..."
Evaluate: Consistent, captures all required info

Phase 4 - Parameter tuning:
Test $T \in \{0.2, 0.3, 0.5\}$ and repetition_penalty $\in \{1.1, 1.2, 1.3\}$
Select: $T = 0.3$, penalty=1.2 based on A/B testing

Prompt engineering is empirical - test on real documents, iterate based on failures

A6: Temperature Scaling Mathematics

Softmax with temperature:

$$P(w_i|context) = \frac{\exp(logit_i / T)}{\sum_j \exp(logit_j / T)}$$

Example: Logits $= [3.0, 2.0, 1.0]$

$T = 0.5$ (peaked):
$P = [\exp(6.0), \exp(4.0), \exp(2.0)]/Z = [0.71, 0.24, 0.05]$

$T = 1.0$ (normal):
$P = [\exp(3.0), \exp(2.0), \exp(1.0)]/Z = [0.58, 0.32, 0.10]$

$T = 2.0$ (flat):
$P = [\exp(1.5), \exp(1.0), \exp(0.5)]/Z = [0.46, 0.31, 0.23]$

Limits:

- $T \to 0$: $P \to$ one-hot (argmax, deterministic)
- $T \to \infty$: $P \to$ uniform (random)

Temperature rescales logits before softmax, controlling distribution sharpness

A7: Nucleus (Top-p) Sampling Algorithm

Algorithm:

1. Compute probabilities: $P(w_1), P(w_2), \ldots, P(w_V)$ via softmax
2. Sort words by probability: $P(w_{(1)}) \geq P(w_{(2)}) \geq \ldots \geq P(w_{(V)})$
3. Compute cumulative sum: $C_k = \sum_{i=1}^{k} P(w_{(i)})$
4. Find cutoff: $k^* = \min\{k : C_k \geq p\}$
5. Sample from top k^* words (renormalize)

Example ($p = 0.9$):

Word	P	C	Include?
"growth"	0.35	0.35	Yes
"increase"	0.25	0.60	Yes
"rise"	0.15	0.75	Yes
"gain"	0.12	0.87	Yes
"surge"	0.08	0.95	Yes
"boost"	0.05	1.00	No

Nucleus size adapts: peaked dist $\rightarrow$ few words, flat dist $\rightarrow$ many words

Top-p is dynamic cutoff, top-k is fixed cutoff (less common)

A8: Repetition Penalty Formulation

Goal: Reduce probability of recently generated tokens

Method 1 - Multiplicative penalty:

$$P'(w_i) = \begin{cases} P(w_i)/\alpha & \text{if } w_i \text{ in recent context} \\ P(w_i) & \text{otherwise} \end{cases}$$

Then renormalize: $P''(w_i) = P'(w_i) / \sum_j P'(w_j)$

Method 2 - Additive penalty (less common):

$$logit_i' = logit_i - \beta \cdot \text{count}(w_i)$$

Typical values: $\alpha \in [1.0, 1.5]$ where:

- $\alpha = 1.0$: No penalty
- $\alpha = 1.1$: Mild (natural variance)
- $\alpha = 1.2$: Moderate (good for summarization)
- $\alpha = 1.5$: Aggressive (may sound unnatural)

Penalty applies to tokens in recent window (e.g., last 64 tokens)

A9: Beam Search for Summarization

Beam search finds high-probability sequences (vs sampling)

Algorithm (beam width k):
Step 0: Start with $[BOS]$ (beginning of sequence)
Step 1: Generate top-k first tokens
Keep k best hypotheses: $H = \{h_1, h_2, \ldots, h_k\}$
Step 2: For each hypothesis h_i, generate all continuations
Score each: $score(h_i + w_j) = \log P(h_i) + \log P(w_j|h_i)$
Keep top-k overall (prune rest)
Step t: Repeat until all beams end or max length
Output: Highest-scoring complete sequence

Length normalization (prevent short bias):

$$score(h) = \frac{1}{|h|^\alpha} \sum_{t=1}^{|h|} \log P(w_t|h_{<t})$$

Typical: $\alpha = 0.6$ (slight penalty for length)

Beam search: deterministic, high quality, no diversity (always same output)

A10: Sampling Strategies Comparison

Method	**How it works**	**Pros**	**Cons**
Greedy	Always pick highest P	Fast, deterministic	Repetitive, no diversity
Pure Sampling	Sample from full P	Diverse	Too random, incoherent
Temperature	Scale logits by T	Simple control knob	Still samples unlikely words if T high
Top-k	Sample from top k words	Fixed vocabulary size	k doesn't adapt to distribution
Top-p (nucleus)	Dynamic cutoff at p	Adapts to peaked/flat	More complex
Beam search	Keep top k hypotheses	High quality, coherent	No diversity, slow

For summarization, typical combination:
Temperature $T = 0.3$ (low randomness) + Top-p $p = 0.9$ (filter unlikely) + Repetition penalty $\alpha = 1.2$ (diversity)

Multiple strategies can be combined (e.g., temperature + top-p + penalty)

A11: Sliding Window for Long Documents

Strategy: Maintain overlapping context between chunks

Algorithm:

1. Split document into chunks of size L tokens
2. Process with overlap O tokens (e.g., $O = 0.2 \cdot L$)
3. Each chunk sees last O tokens from previous chunk
4. Prevents loss of context at boundaries

Example ($L = 1000$, $O = 200$):
Chunk 1: tokens [0, 1000]
Chunk 2: tokens [800, 1800] (overlaps 800-1000)
Chunk 3: tokens [1600, 2600] (overlaps 1600-1800)

Benefit: Sentences spanning chunk boundaries are fully captured
Cost: Process O tokens twice (1.2x total tokens for $O = 0.2L$)

Overlap essential for maintaining coherence across chunks

A12: Hierarchical Merging Strategy

Recursive summarization preserves document structure

Full algorithm:
Level 0 (base): Summarize each section independently
$S_1, S_2, \ldots, S_n \rightarrow$ summaries $s_1, s_2, \ldots, s_n$
Level 1: Group related summaries, merge
$(s_1, s_2) \rightarrow s_{12}$, $(s_3, s_4, s_5) \rightarrow s_{345}$, etc.
Level 2: Merge Level 1 summaries
$(s_{12}, s_{345}) \rightarrow s_{final}$

Grouping strategies:

- By document structure (Introduction + Methods, All Results, Discussion)
- By topic (cluster similar sections)
- Fixed size (every k sections)

Advantages:

- Preserves logical document flow
- Maintains topic coherence within groups
- Reduces redundancy (each fact summarized once per level)

Hierarchical ¿ flat chunking for structured documents (papers, reports)

A13: Attention Sink and Context Management

Challenge: LLMs have limited attention to very distant tokens

Attention patterns in long contexts:

- **Recency bias**: Attend more to recent tokens
- **Attention sink**: First few tokens get disproportionate attention
- **Middle loss**: Tokens in middle of long context often ignored

Implication for summarization:
Placing document at different positions affects summary quality:

- Position 1 (after prompt): Gets attention sink benefit
- Position middle: May be partially ignored
- Position end: Gets recency benefit

Best practices:

- Keep prompts short (save tokens for document)
- Place most important content early or late in chunk
- For multi-chunk: Repeat critical info (e.g., key definitions) in each chunk prompt

Context position matters: beginning and end attended more than middle

Task: Summarize 10-100 related documents into one coherent summary

Challenges:

- Identify common themes vs unique points
- Avoid redundancy (same fact mentioned in many docs)
- Maintain attribution (which doc said what)
- Handle contradictions between sources

Approach 1 - Map-Reduce with deduplication:
Map: Summarize each document $\rightarrow s_1, \ldots, s_n$
Deduplicate: Cluster similar sentences, keep one per cluster
Reduce: Merge deduplicated summaries $\rightarrow$ final summary

Approach 2 - Query-focused:
Prompt: "Given these 10 articles about climate policy, summarize: (1) consensus findings, (2) disagreements, (3) policy recommendations mentioned"
Forces analysis across documents

Approach 3 - Hierarchical by topic:
Cluster documents by topic $\rightarrow$ Summarize each cluster $\rightarrow$ Merge cluster summaries

Multi-document harder than single-document due to redundancy and contradictions

A15: Production System Considerations

Deploying LLM summarization at scale:

Latency:

- 1-3 seconds per summary (typical)
- Batch processing for non-urgent use cases
- Caching for repeated documents

Cost:

- GPT-4: $0.03 per 1K input tokens, $0.06 per 1K output
- For 5K input + 200 output: $0.16 per summary
- Use cheaper models (GPT-3.5, open-source) when possible
- Test cost vs quality tradeoff

Quality control:

- Sample 1% for human evaluation
- Automated checks: length, formatting, profanity filter
- Hallucination detection (faithfulness to source)
- Fallback to extractive if LLM fails

Monitoring:

Lab Implementation Details

Code-Level Walkthrough — Real Outputs — Hands-On Concepts

A16: Lab Overview - What We Implemented

4-Part Lab Structure:

Part 1: Setup and Model Loading

- Load FLAN-T5-small via Hugging Face Transformers
- Configure device (CPU/GPU)

Part 2: Prompt Engineering Experiments

- Zero-shot vs few-shot comparison
- Same article, different prompts

Part 3: Decoding Parameter Experiments

- Temperature: 0.3, 0.7, 1.0
- Top-p: 0.8, 0.9, 0.95
- Repetition penalty: 1.0, 1.2, 1.5

Part 4: Long Document Handling

- Chunking strategy with overlap
- Merge chunk summaries

Total: 19 cells, runs on CPU (10 min) or GPU (2 min)

A17: FLAN-T5 Model Loading Code

Why FLAN-T5-small?

- **Size**: 80M parameters (fits on CPU)
- **Speed**: Fast inference (1-2 sec/summary on CPU)
- **Quality**: Instruction-tuned, good for summarization

Loading Code:

```
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM
import torch

model_name = "google/flan-t5-small"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSeq2SeqLM.from_pretrained(model_name)

device = "cuda" if torch.cuda.is_available() else "cpu"
model = model.to(device)
print(f"Model␣loaded␣on␣{device}")
```

Real Output:

```
PyTorch version: 2.5.1+cpu
CUDA available: False
Model loaded on cpu
```

AutoModelForSeq2SeqLM: Encoder-decoder architecture for text-to-text tasks

A18: Model Comparison - FLAN-T5 Variants

Model	Parameters	Memory	Speed	Quality
flan-t5-small	80M	300MB	Fast (2s)	Good
flan-t5-base	250M	1GB	Medium (5s)	Better
flan-t5-large	780M	3GB	Slow (15s)	Best
flan-t5-xl	3B	11GB	Very slow (60s)	Excellent

Hardware Requirements:

- **CPU**: Works for small/base (8GB+ RAM recommended)
- **GPU**: Recommended for large/xl (16GB+ VRAM)
- **Cloud**: Use Google Colab (free T4 GPU) or AWS

Speed vs Quality Trade-off:

- Development: Use small (fast iteration)
- Production: Test base vs large (quality matters)
- Research: Use xl if available (best results)

Lab uses small for accessibility - works on any laptop

A19: Tokenizer Mechanics Code

Tokenization Process:

```
# Input: raw text string
text = "Summarize:␣The␣Fed␣raised␣interest␣rates..."

# Tokenizer converts to model inputs
inputs = tokenizer(
    text,
    return_tensors="pt",    # PyTorch tensors
    max_length=512,         # Truncate if longer
    truncation=True         # Enable truncation
).to(device)

# Output: dictionary with input_ids and attention_mask
print(inputs.keys())  # dict_keys(['input_ids', 'attention_mask'])
print(inputs['input_ids'].shape)  # torch.Size([1, N])
```

What Happens:

1. Text split into subword tokens (SentencePiece)
2. Each token mapped to integer ID
3. IDs converted to PyTorch tensor
4. Attention mask created (1=real token, 0=padding)

---

return_tensors="pt" returns PyTorch tensors (vs "tf" for TensorFlow)

A20: Special Tokens and Truncation

FLAN-T5 Special Tokens:

- **PAD** (0): Padding token (unused in seq2seq generation)
- **EOS** (1): End-of-sequence (marks end of output)
- **UNK** (2): Unknown token (rare words)

512 Token Limit:
Input: "Summarize: [1000-word article]"
Token count: $\sim$250 tokens
Problem: If article + prompt $>$ 512 tokens $\rightarrow$ truncation
Solution:

- Truncate input (`truncation=True`)
- OR use chunking strategy (Part 4)

Real Example:
Article: 160 words = $\sim$200 tokens
Prompt: "Summarize this article in 3 sentences" = $\sim$10 tokens
Total: $\sim$210 tokens (well under 512 limit)

1 token $\approx$ 0.75 words (English text). Context window = max input length

A21: Generate() Function Parameters

Complete Generation Code:

```
outputs = model.generate(
    **inputs,                        # Unpacked input_ids, attention_mask
    max_length=100,                  # Max output tokens (not words)
    temperature=0.7,                 # Randomness (0=deterministic, 2=chaos)
    top_p=0.9,                       # Nucleus sampling cutoff
    repetition_penalty=1.2,          # Penalize repeated tokens (>1.0)
    do_sample=True,                  # Use sampling (False=greedy)
    num_return_sequences=1           # Number of outputs to generate
)

# Decode output tokens back to text
summary = tokenizer.decode(outputs[0], skip_special_tokens=True)
```

Parameter Types:

- **Length**: max_length (int)
- **Randomness**: temperature (float), do_sample (bool)
- **Filtering**: top_p (float 0-1), top_k (int)
- **Penalties**: repetition_penalty (float $\geq$ 1.0)
- **Batch**: num_return_sequences (int)

do_sample=True required for temperature/top-p to take effect

A22: Decoding Parameter Effects (Real Outputs)

Same Article, Different Parameters:

Setting	Actual Output from Notebook
T=0.3 (factual)	"Federal Reserve chiefs have raised interest rates to a range of 5.00% to 5.25%, the highest level in 16 years."
T=0.7 (balanced)	"Federal Reserve President Mark Zuckerberg told the Wall Street Journal the Federal Reserve remained calm in the wake of the flurry of interest rates."
T=1.0 (creative)	"Federal Reserve chair Jerome Powell said the US rate had been lowered, a move which highlights ongoing uncertainty as the central bank faces interest rates."

Top-p=0.8	"Federal Reserve officials have raised interest rates by 0.25 percentage points in a bid to cut interest rates, despite a decline in inflation."
Top-p=0.9	"Federal Reserve officials say they will monitor data on a possible rate hike to keep inflation lower."
Top-p=0.95	"Federal Reserve Chairman Jerome Powell said he would monitor the current rate growth rate..."

Observation: Lower temperature (0.3) gives most accurate summary. Higher values introduce errors (e.g., "Mark Zuckerberg").

For summarization: T=0.3-0.5, p=0.9, penalty=1.2 work best

A23: Optimal Parameter Combination

Best Practices from Lab:

```
# Optimal configuration for factual summarization
outputs = model.generate(
    **inputs,
    max_length=100,          # Allow enough space for summary
    temperature=0.3,         # Low randomness = factual
    top_p=0.9,              # Filter bottom 10% unlikely words
    repetition_penalty=1.2,  # Mild penalty for variety
    do_sample=True           # Enable sampling
)
```

Why These Values:

- **temperature=0.3**: Factual accuracy ¿ creativity
- **top_p=0.9**: Remove very unlikely words, keep reasonable options
- **repetition_penalty=1.2**: Avoid "the company... the company..." but not too aggressive
- **max_length=100**: Typical summary length (50-100 tokens = 30-75 words)

Real Output with Optimal Settings:
`''Federal Reserve chiefs have raised interest rates to a range of 5.00% to 5.25%, the highest level in 16 years.''`
Quality: Accurate, concise, no hallucinations

Always test parameter combinations on your specific domain/task

A24: Chunking Algorithm Implementation

Problem: Document too long for 512 token limit
Solution: Split into overlapping chunks

Full Implementation:

```
def chunk_text(text, chunk_size=500, overlap=100):
    """Split␣text␣into␣overlapping␣chunks␣by␣words"""
    words = text.split()  # Split by whitespace
    chunks = []

    # Step through text with stride = chunk_size - overlap
    for i in range(0, len(words), chunk_size - overlap):
        chunk = "␣".join(words[i:i + chunk_size])
        if chunk:  # Only add non-empty chunks
            chunks.append(chunk)

    return chunks

# Example: 800-word document
chunks = chunk_text(long_doc, chunk_size=300, overlap=50)
# Result: 3 chunks of ~300 words each
# Chunk 1: words 0-299
# Chunk 2: words 250-549 (50-word overlap with chunk 1)
# Chunk 3: words 500-799 (50-word overlap with chunk 2)
```

Overlap ensures sentences spanning boundaries are captured in at least one chunk

A25: Chunking Example with Real Outputs

Input: 5x repeated article = 800 words

Chunking Parameters: chunk_size=300, overlap=50

Result: 3 chunks created

- Chunk 1: 300 words
- Chunk 2: 300 words (overlaps with chunk 1 by 50 words)
- Chunk 3: 250 words (remaining text)

Processing Strategy:
Step 1: Summarize each chunk independently
Chunk 1 → Summary 1 (50 tokens)
Chunk 2 → Summary 2 (50 tokens)
Chunk 3 → Summary 3 (50 tokens)

Step 2: Combine all summaries
Combined text: Summary 1 + Summary 2 + Summary 3 = 150 tokens

Step 3: Summarize the summaries
Final summary: 80 tokens (comprehensive overview)

Key Insight: Hierarchical summarization preserves information better than single-pass truncation

This is the "map-reduce" strategy discussed in main presentation