Fine-Tuning Language Models & Managing Large Codebases in Limited Context Windows

By: Khushi Malik

My previous research:https://github.com/khushimalik3122/llm_research/tree/main

---

Part 1: Fine-Tuning Language Models

Aspect	Explanation	Example
What is Fine-Tuning?	Training a pre-trained LLM further on a specific dataset.	Training GPT-4 on legal documents to make it a legal assistant.
Why Fine-Tune?	To make the model better at specific tasks or domains.	A fine-tuned model can summarize patient notes better than a general one.
Instruction-Tuning	A type of fine-tuning where the model learns to follow instructions.	Instruction: "Translate to Spanish" → Input: "Hello" → Output: "Hola"
Transfer Learning	Uses what the model already knows to learn new tasks with less data.	Uses language knowledge from pretraining to learn medical summaries quickly.
Trade-Off: Specialization vs. Generalization	Fine-tuning may hurt general abilities (catastrophic forgetting).	A medical model may forget how to write fiction or answer trivia.
Behavior Adaptation	Fine-tuning changes not just facts, but how the model reasons.	A model trained with QLoRA on medical data can suggest treatments.
Small Data, Big Impact	Fine-tuning doesn't need as much data as pretraining, thanks to transfer learning.	Just thousands of labeled examples can significantly improve performance.

Full Fine-Tuning vs. Parameter-Efficient Fine-Tuning (PEFT)

Method	How It Works	Pros	Cons	Best Use Cases
Full Fine-Tuning	Updates all model parameters using a new dataset.	- Maximum performance- Complete control over behavior	- Very expensive (requires powerful GPUs and time)- Not scalable for multiple tasks- Risk of forgetting old skills	Building a single, highly-specialized copilot for one domain (e.g., bioinformatics code generation)
LoRA (Low-Rank Adaptation)	Freezes base model and adds small trainable low-rank matrices inside layers.	- Much cheaper than full fine-tuning- High performance with limited compute	- May forget pretraining knowledge- Switching tasks is slow due to cache recomputation	Adding domain-specific skills (e.g., ReactJS patterns, API generation) to your coding assistant
QLoRA	Combines LoRA with 4-bit quantization for lower memory usage.	- Can fine-tune large models on standard GPUs- Maintains high performance- Low cost	- Requires quantization setup- May still forget general knowledge	Building affordable copilots for startups or small teams (e.g., local development assistants)
Adapter Tuning	Adds small, trainable modules between Transformer layers; base model stays frozen.	- Only a small percentage of the model is trained- Supports continual learning- Reduces forgetting	- May offer less flexibility for deeply changing behavior	Training a copilot across multiple languages or domains over time without starting from scratch

Concern	Insight
Forgetting	LoRA and QLoRA can forget general skills if not properly tuned. Adapter tuning is more robust for continual learning.
Cost	Full fine-tuning is expensive. PEFT methods like QLoRA make fine-tuning accessible for smaller teams and individuals.
Efficiency	PEFT allows reuse of the base model and separate modules per task, making it ideal for multi-skill copilots.
Continual Learning	Adapter tuning is best when the copilot needs to be updated regularly with new skills or domains.

When to Fine-Tune vs. Prompt Engineer or Use RAG

Method	How It Works	Pros	Cons	How to Handle Cons (Coding Copilot)
Prompt Engineering	Carefully designing prompts or few-shot examples to guide the LLM without changing its weights.	- Fast to implement- No training needed- Cost-effective	- Limited control- Inconsistent results- Poor for complex logic/code generation	- Use structured prompts/templates- Combine with few-shot examples- Escalate to RAG/fine-tuning for complex tasks

RAG(Retrieval-Augmented Generation)	Retrieves relevant context from external sources and feeds it into the LLM during inference.	- Access to real-time/project-specific info- Reduces hallucination- Transparent (can cite sources)	- Depends on retrieval quality- Can struggle with code structure & state- Slower due to extra retrieval step	- Use smart chunking (e.g., function-level)- Optimize embeddings for code- Cache frequent queries- Use LLMs that are trained to handle RAG context
Fine-Tuning	Updates the LLM's internal parameters using labeled training data.	- Deep task adaptation- Precise code formatting or reasoning- Ideal for instruction-following	- Expensive- Needs labeled data- Risk of overfitting or forgetting	- Use PEFT methods (e.g., QLoRA)- Train with diverse, real-world code tasks- Combine with RAG for dynamic knowledge

Real-World Challenges in Fine-Tuning Large Models

Challenge	Impact	Mitigation Strategies	Best Practices for Coding Copilot
Compute Cost	- Very high GPU/memory requirements- Long training times- Expensive cloud bills	- Use PEFT methods (LoRA, QLoRA)- Mixed-precision (16-bit/8-bit) training- Spot instances & off-peak scheduling- Smaller task-specific models	- Use **QLoRA** to fine-tune on coding data using a single 24GB GPU- Train smaller language models for different coding domains (e.g., frontend vs backend)
Catastrophic Forgetting	- Model forgets general capabilities after task-specific tuning- Reduces overall versatility	- Mix general data with domain data (rehearsal)- Use synthetic SFT-like data- Apply adapter scaling or model merging- Experience replay	- Fine-tune using a mix of coding tasks (e.g., data science, DevOps, frontend)- Use adapters or LoRA modules per task, with careful merging for final deployment
Data Quality	- Poor-quality data leads to hallucinations, bugs, and bias- High annotation costs and legal concerns	- Rigorous data filtering and deduplication- Use trusted sources (open-source repos, docs)- Data augmentation & labeling standards- MLOps pipelines with freshness checks	- Curate clean, labeled code snippets (e.g., from GitHub stars >100)- Set up auto-refresh pipelines with linted and tested code data- Add timestamp/version control for code compatibility

Part 2: Fitting Large Codebases in Context Window

Aspect	Details
What is a Context Window?	The maximum number of tokens an LLM can process in one input. It acts like the model's short-term memory.
Early Limits	GPT-2/GPT-3: 2K–4K tokens → only handled short text (few pages).
Modern Advances	- GPT-4 (2023): 32K tokens- GPT-4.1 (2025): 1M tokens- Claude 2/3: Up to 100K tokens- Gemini 1.5: 2M tokens- LLaMA 4 Scout: Pushing toward 10M tokens
Why It Matters	Long context enables tasks like summarizing legal documents, analyzing full codebases, or multi-turn conversations without forgetting earlier input.
Key Challenges	- Quadratic compute ($O(n^2)$) → higher cost as tokens increase- Performance drops beyond trained length- Models struggle with long-range reasoning and needle-in-haystack tasks
Technical Constraints	- Longer context = more memory + slower response- Many models weren't trained on long-context data- Perplexity may spike when exceeding pre-trained context
Solutions in Progress	- Sparse Attention- Sliding Windows- Hierarchical Memory- Efficient Transformers (e.g., Longformer, FlashAttention)
Best Practices (Coding Copilot)	- Keep relevant code snippets close together in the prompt- Use retrieval (RAG) to inject only the needed functions/files- Trim unnecessary tokens (comments, logs)- Use models fine-tuned for long-context (e.g., Claude, Gemini, GPT-4.1)
Key Insight	Bigger context windows don't guarantee better performance—attention efficiency and data quality still drive output quality.

For Coding Copilots: Strategy to Handle Long Contexts

Problem	Suggested Strategy
Code too long for context window	Use RAG to fetch only relevant chunks (e.g., functions, imports, class defs)
Model misses critical code details	Implement context-aware ranking of retrieved code (prioritize nearest calls or relevant dependencies)
Model slows down on large files	Use models with sparse or sliding attention and token-pruning techniques
Forgetting previous code interactions	Integrate conversation memory or session history summarization

Techniques to Fit Large Codebases in Context Window

Technique	Description	Pros	Challenges	Use in Coding Copilots
1. Chunking	Splitting code into smaller parts	Enables fitting into context	May split logic mid-function	Use AST-based or semantic chunking to preserve structure
- Naïve Chunking	Fixed-size splits (e.g., every 256 tokens)	Simple & fast	Breaks syntax/logic	Avoid for complex tasks
- AST-Based Chunking	Splits by function/class/module using parsers	Preserves logic & syntax	Requires language-specific tools	Recommended default for code ingestion
- Semantic Chunking	Splits based on topic shifts using NLP	Coherent, meaningful chunks	Hard to implement in code	Combine with AST for better results

- LLM/Expert Chunking	Manual or LLM-guided splits	Human-aligned logic	Time-consuming or costly	Use for critical sections or new projects
2. Sliding Window & Overlapping	Adds overlapping tokens between chunks	Retains flow across chunks	Increases total token usage	Use chunk_overlap of 10–20% for continuity
3. RAG Pipelines	Fetches relevant code from a vector DB	Dynamically injects relevant code	Needs good chunking + retrieval quality	Core for copilots handling large projects
4. Hierarchical Attention	Mimics memory layers (short/long-term)	Handles massive input sizes	Needs advanced model support	Use with custom LLM backends for better recall
5. Vector Databases	Stores embedded code for semantic search	Fast and scalable	ANN search tuning is critical	Use tools like FAISS, Weaviate, or Pinecone
6. Summarization & Code Folding	Summarizes functions or files to reduce tokens	Improves comprehension, saves space	Risk of info loss	Summarize less-relevant sections in prompts
7. File-Tree Aware Prompting	Includes project structure and file paths	Preserves hierarchy & relationships	Needs formatting conventions	Use @workspace, #filename tags for scoped queries
8. Code Graphs (Emerging)	Graph-based code representation	Captures deep semantic dependencies	Early-stage & complex	Future copilots may use this for multi-hop reasoning

Recommended Strategy for Coding Copilots

Goal	Approach
Fit large codebase in prompt	Use AST-based chunking + overlap (10–20%)
Provide relevant code on request	Implement RAG pipeline with a vector DB
Maintain logic across files	Add file tree references or structure-aware tags
Avoid hallucinations	Retrieve verified internal code + documentation
Speed up response	Cache frequently used embeddings + summaries
Improve answer precision	Summarize folded code or large files before sending
Handle ambiguous queries	Use vector search + file structure to scope intent

Part 3: Limitations & Research Gaps

Challenge	Description	Impact	Suggested Copilot Strategy
1. Token Limits & Long-Range Dependency Loss	Even with million-token windows, LLMs struggle to connect distant code elements (e.g., helper functions or dependencies defined far apart).	Causes reasoning gaps in large codebases; reduced accuracy in tasks requiring global context.	Use **hierarchical memory** or **agent-based chunking**; allow model to request additional context on demand.
2. RAG Limitations for Code	RAG fails to fully capture inter-file relationships, global state, and logical flow in repositories; retrieval based on surface similarity misses structural intent.	Leads to incomplete responses, errors in cross-module understanding.	Enhance RAG with **file-tree awareness**, **dependency graphs**, and **structured prompts**. Consider **code-specific retrievers** instead of generic embeddings.
3. Semantic Loss from Plain Chunking	Naïve chunking breaks code logic (e.g., splitting functions mid-body), losing syntactic and semantic coherence.	Reduces effectiveness of retrieval and reasoning; incoherent or partial responses.	Prefer **AST-based chunking** + LLM-assisted logical partitioning. Track metadata (e.g., class/function/file scope).
4. Evaluation Deficiencies in Long-Context Tasks	Existing benchmarks fail to capture real-world complexity (cross-file logic, debugging, repair, etc.). Accuracy sharply drops with longer contexts.	Inflated performance assumptions; poor alignment with user expectations in complex coding tasks.	Incorporate real-world test cases. Use **LongCodeBench** or create domain-specific long-context benchmarks. Track reasoning chains, not just exact matches.

5. Syntax-Semantics Gap	LLMs often process code as plain text, failing to internalize its executable logic or structural constraints.	Misunderstanding of type systems, data flow, inheritance; hallucinations of invalid patterns.	Embed **code structure as graphs**, not just text (e.g., function call graphs, ASTs). Use **symbol-aware** or **type-aware LLM backends**.
6. Passive Context Consumption	Current LLMs consume context passively rather than managing it actively; they cannot fetch, prioritize, or summarize context themselves.	Limits scalability and leads to inefficient memory use; requires manual prompt engineering.	Move toward **agentic copilots**: models that can request, cache, and prune context based on task need and codebase state. Implement **memory modules + planning logic**.

Summary of Best Practices

When to Use Fine-Tuning vs. Smarter Prompting/RAG

Approach	Use Cases	Benefits	Copilot Implementation Tips
Fine-Tuning	- Deep domain expertise (e.g., medical, legal)- High-accuracy code generation- Instruction or safety alignment	- Internalizes domain knowledge- Consistent outputs- Custom behavior control	Use **QLoRA** for low-cost adaptationStart with **small models (7B)** if resources are constrained
Prompting / RAG	- Real-time knowledge needs- Rapid prototyping- Resource-limited environments- Traceable answers	- No training required- Transparent outputs- Fast iteration	Use **smart prompts**, few-shot examplesIntegrate **vector database for RAG** with metadata

Technique	Description	Best Practice
Code-Aware Chunking	Break code by structure (function, class, module)	Use **ASTs** or LLM-guided chunking to preserve logic
Overlapping Windows	Include slight overlap between chunks	Use 10-20% overlap to retain cross-boundary context
Metadata Enrichment	Add file paths, function names, module types to each chunk	Store as metadata fields in the vector DB
Hierarchical Prompting	Reflect codebase architecture in prompts (tree, file paths)	Implement **file-aware layouts** or structured prompts
Hybrid Retrieval	Combine semantic + keyword + reranking	Avoid retrieval gaps from relying on a single method
Iterative Refinement	Continuously test & evolve your strategy	A/B test retrieval quality, chunk granularity, overlap ratio

The table below provides a structured decision matrix to guide the choice between fine-tuning, prompt engineering, and RAG, incorporating key considerations and relevant code context management techniques.

Best Practices Decision Matrix

Strategy	Best For	Key Considerations	Code Context Management Techniques
Fine-Tuning	Deep Domain Specialization, Superior Performance, Behavioral Alignment	High Compute Cost, Catastrophic Forgetting, Data Quality	PEFT (LoRA, QLoRA), Data Curation & Augmentation, Rehearsal Methods
Prompt Engineering	Rapid Prototyping, General Tasks, Cost-Sensitive Scenarios	Performance Limitations, Lack of Deep Adaptation	Structured Prompts, In-context Learning (Few-shot)
Retrieval-Augmented Generation (RAG)	Dynamic/Fresh Knowledge, Factual Accuracy, Domain-Specific Data (without retraining)	Retrieval Quality, Latency, Cross-File References, Global State	Code-Aware Chunking, Overlapping Chunks, Vector Databases, Re-rankers

Emerging Methods & Future Outlook

Method	What It Is	Benefits	Relevance to Code Copilots	Implementation Notes
Mixture-of-Experts (MoE)	An architecture that activates only a subset of model parameters ("experts") per input	- High capacity with lower compute cost- Efficient inference- Modular reasoning	Enables scalable understanding of diverse code types, languages, or domains	Can specialize experts for front-end, back-end, testing, etc.
Persistent Memory	Embeds a long-term memory pool in the LLM that updates over time	- Continuously learns from evolving codebases- Retains long-term context- Bypasses static training limitations	Maintains awareness of codebase history, changes, and team coding patterns	Ideal for live projects with frequent code updates
Flash Attention 2	Optimized attention algorithm to reduce compute/memory overhead in transformers	- 2x speedup over v1- 55% less GPU memory usage- Enables long-context processing	Makes it practical to feed entire files or multiple files into the model at once	Supports more efficient large-context Copilot usage with lower latency

Coding Copilot: Practical Context Management in Action

Technique	What It Does	Practical Benefits	Typical Usage Example
Slash Commands (/doc, /explain, /fix, /optimize, /tests, etc.)	Let the user state intent and task scope in one token.	Fast, unambiguous task setup; lowers prompt-engineering friction.	/fix selected code block to auto-suggest a patch.
File / Symbol References (#File.cs, #MethodName, #File.cs:66-72)	Target exact files, functions, or line ranges.	Reduces noise; LLM sees only relevant code.	#AddItemToBasket to focus suggestions on that method.
Workspace Awareness (@workspace, @github)	Gives the LLM visibility over the whole project (or repo).	Global context for cross-file refactors, dependency fixes.	@workspace then "Suggest type-safe migration from v1 to v2".
Output-Log Referencing	Paste build/runtime logs into the prompt.	Enables error triage and debug explanations.	Insert test failure stack trace, ask "Why is this flaking?"
Prompt Files / Templates	Pre-saved, well-crafted prompts for recurring tasks.	Consistent instructions; team-wide standardization.	prompt:generate-rest-endpoint.txt to scaffold endpoints.
Visual Inputs	Attach UI screenshots or diagrams.	UI-to-code, layout fixes, accessibility checks.	Drop a screenshot; ask for matching React component.
Context Summarization / Trimming	Auto-compress long chat history when nearing token limit.	Keeps long sessions coherent without manual pruning.	Claude auto-summarizes prior steps after ~100 KB context.
Multi-Agent Back-End (implicit)	Sub-agents tackle sub-tasks in isolated windows.	Parallel, focused reasoning for complex jobs.	One agent parses logs; another writes unit tests; results merged.

How to Apply These Ideas to *Your* Copilot

1. **Adopt clear slash commands** so users can declare intent in a single word.
2. **Expose file/symbol selectors** in the IDE to feed only pertinent code chunks.
3. **Maintain a workspace index** (file tree + embeddings) so the model can answer repo-wide questions.
4. **Accept runtime artifacts** (logs, traces) as first-class context objects.
5. **Ship prompt templates** for repetitive chores—unit-test stubs, docstrings, optimizations.
6. **Auto-summarize long chats** to stay within context limits while preserving key decisions.
7. **Consider a lightweight agent layer** if tasks often decompose into independent subtasks (e.g., linting, refactor, doc gen).

The table below illustrates practical examples of how GitHub Copilot utilizes various context management techniques.

GitHub Copilot Context Management Examples

Command/Reference	Usage Example	Context Provided to LLM
/explain	/explain the AddItemToBasket method in BasketService.cs	AddItemToBasket method in BasketService.cs
#MyFile.cs	Where are the tests in #BasketService.cs?	Entire BasketService.cs file
#MyFile.cs: 66-72	What is the purpose of #MyFile.cs: 66-72?	Exact code section from lines 66-72 of MyFile.cs
@workspace	Is there a delete basket method in this @workspace	Current solution open in the IDE (all open files, projects, configurations)
#output	Check my output logs and help me fix this error.	Content of the active output logs (e.g., build errors)
/doc	select desired code and enter /doc	Selected code snippet

Conclusions

Focus Area	Key Insight	Practical Takeaway
Fine-Tuning	PEFT (LoRA, QLoRA) enables deep domain adaptation at reduced compute cost.	Use for domain-heavy tasks or when behavior consistency is critical (e.g., internal tooling, compliance-sensitive code).
Fine-Tuning Risk	Catastrophic forgetting remains a key issue.	Mitigate using rehearsal techniques, adapter scaling, and memory replay strategies.
Prompting & RAG	RAG fills external knowledge gaps dynamically, with less training cost.	Best for general-purpose tools, real-time info, and limited-resource environments. Combine with fine-tuning for hybrid solutions.
Chunking Code	Code requires **structure-aware** chunking (AST, function-level, semantic).	Avoid fixed-size splits. Use LLM-aware partitioning or AST parsers to preserve logic.
Long Context Use	Larger context windows ≠ better understanding. Performance often drops with length.	Focus on selective summarization, overlap strategies, and structured memory use.
Context Engineering	Intelligent pre-processing (chunk overlap, metadata, vector search) is essential.	Treat context as a design artifact, not just a size constraint.
Knowledge Graphs for Code	Trend: move from flat text to graphs capturing code structure and dependencies.	Enables reasoning over imports, class hierarchies, and architectural patterns.
Agentic LLM Behavior	Future LLMs will *manage their own context*: select, summarize, retrieve, and adapt.	Develop LLMs as **active agents**, not passive engines.
Emerging Innovations	MoE, Persistent Memory, Flash Attention 2 → better compute scaling, continual learning, and faster inference.	Invest in or monitor these for long-horizon capabilities in live, evolving codebases.
Future of LLMs for Code	LLMs are evolving into dynamic assistants that can grow with a codebase.	Anticipate tools that understand full project histories and architect-level abstractions.

Works cited

1. Fine-Tuning Large Language Models for Specialized Use Cases - PMC, accessed July 5, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC11976015/
2. Prompt Engineering or Fine-Tuning: An Empirical Assessment of LLMs for Code - arXiv, accessed July 5, 2025, https://arxiv.org/html/2310.10508v2
3. arXiv:2502.08130v2 [cs.CL] 20 Feb 2025, accessed July 5, 2025, https://arxiv.org/pdf/2502.08130
4. Improved Supervised Fine-Tuning for Large Language Models to Mitigate Catastrophic Forgetting - arXiv, accessed July 5, 2025, https://arxiv.org/html/2506.09428v2
5. Lightweight Clinical Decision Support System using QLoRA-Fine-Tuned LLMs and Retrieval-Augmented Generation - arXiv, accessed July 5, 2025, https://arxiv.org/html/2505.03406v1
6. LoRA vs Full Fine-tuning: An Illusion of Equivalence - arXiv, accessed July 5, 2025, https://arxiv.org/html/2410.21228v2
7. LoRA vs Full Fine-tuning: An Illusion of Equivalence - arXiv, accessed July 5, 2025, https://arxiv.org/html/2410.21228v1
8. What is the cost of training large language models? - CUDO Compute, accessed July 5, 2025, https://www.cudocompute.com/blog/what-is-the-cost-of-training-large-language-models
9. What is the cost of fine-tuning LLMs? | by The Educative Team | Jul, 2025 - Medium, accessed July 5, 2025, https://medium.com/educative/what-is-the-cost-of-fine-tuning-llms-f5801c00b06d
10. Parameter-Efficient Transfer Learning for NLP - arXiv, accessed July 5, 2025, https://arxiv.org/pdf/1902.00751
11. Adapter-X: A Novel General Parameter-Efficient Fine-Tuning Framework for Vision - arXiv, accessed July 5, 2025, https://arxiv.org/html/2406.03051v2
12. Activated LoRA: Fine-tuned LLMs for Intrinsics - arXiv, accessed July 5, 2025, https://arxiv.org/html/2504.12397v3
13. QLORA: Efficient Finetuning of Quantized LLMs - arXiv, accessed July 5, 2025, https://arxiv.org/abs/2305.14314
14. A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications - arXiv, accessed July 5, 2025, https://arxiv.org/html/2402.07927v2
15. Retrieval-Augmented Generation for Large Language ... - arXiv, accessed July 5, 2025, https://arxiv.org/abs/2312.10997
16. Get Experience from Practice: LLM Agents with Record & Replay - arXiv, accessed July 5, 2025, https://arxiv.org/html/2505.17716v1
17. Common RAG challenges in the wild and how to solve them | by The Educative Team, accessed July 5, 2025,

https://learningdaily.dev/common-rag-challenges-in-the-wild-and-how-to-solve-them-5713bd7ad35c
18. CodeRAG: Supportive Code Retrieval on Bigraph for Real-World Code Generation - arXiv, accessed July 5, 2025, https://arxiv.org/html/2504.10046v1
19. 5 Problems Encountered Fine-Tuning LLMs with Solutions - MachineLearningMastery.com, accessed July 5, 2025, https://machinelearningmastery.com/5-problems-encountered-fine-tuning-llms-with-solutions/
20. Mitigating Catastrophic Forgetting in Continual Learning Using the ..., accessed July 5, 2025, https://thesai.org/Downloads/Volume16No4/Paper_14-Mitigating_Catastrophic_Forgetting_in_Continual_Learning.pdf
21. Improved Supervised Fine-Tuning for Large Language Models to Mitigate Catastrophic Forgetting - arXiv, accessed July 5, 2025, https://arxiv.org/html/2506.09428v1
22. Analyzing Mitigation Strategies for Catastrophic Forgetting in End-to-End Training of Spoken Language Models - arXiv, accessed July 5, 2025, http://arxiv.org/pdf/2505.17496
23. LLM Data Quality: Old School Problems, Brand New Challenges - Gable.ai, accessed July 5, 2025, https://www.gable.ai/blog/llm-data-quality
24. Long-Context Windows in Large Language Models: Applications in ..., accessed July 5, 2025, https://medium.com/@adnanmasood/long-context-windows-in-large-language-models-applications-in-comprehension-and-code-03bf4027066f
25. Context Engineering - LangChain Blog, accessed July 5, 2025, https://blog.langchain.com/context-engineering-for-agents/
26. LLM Maybe LongLM: SelfExtend LLM Context Window Without Tuning - arXiv, accessed July 5, 2025, https://arxiv.org/html/2401.01325v3
27. Extending Language Model Context Up to 3 Million Tokens on a Single GPU - arXiv, accessed July 5, 2025, https://arxiv.org/html/2502.08910v1
28. Sliding Window Attention Training for Efficient Large Language ..., accessed July 5, 2025, https://arxiv.org/abs/2502.18845
29. Xnhyacinth/Awesome-LLM-Long-Context-Modeling: Must-read papers and blogs on LLM based Long Context Modeling - GitHub, accessed July 5, 2025, https://github.com/Xnhyacinth/Awesome-LLM-Long-Context-Modeling
30. Evaluating Long Range Dependency Handling in Code Generation LLMs - arXiv, accessed July 5, 2025, https://arxiv.org/html/2407.21049v2
31. LongCodeBench: Evaluating Coding LLMs at 1M Context Windows - arXiv, accessed July 5, 2025, https://arxiv.org/html/2505.07897v2
32. Can LLMs Replace Humans During Code Chunking? - arXiv, accessed July 5, 2025, https://arxiv.org/html/2506.19897v1
33. Optimizing Chunking, Embedding, and Vectorization for Retrieval ..., accessed July 5, 2025, https://medium.com/@adnanmasood/optimizing-chunking-embedding-and-vectorization-for-retrieval-augmented-generation-ea3b083b68f7

34. Chunking Strategy for LLM Application: Everything You Need to Know - AIVeda, accessed July 5, 2025, https://aiveda.io/blog/chunking-strategy-for-llm-application
35. Chunking strategies for RAG tutorial using Granite - IBM, accessed July 5, 2025, https://www.ibm.com/think/tutorials/chunking-strategies-for-rag-with-langchain-watsonx-ai
36. MoC: Mixtures of Text Chunking Learners for Retrieval-Augmented Generation System, accessed July 5, 2025, https://arxiv.org/html/2503.09600v1
37. Vision-Guided Chunking Is All You Need: Enhancing RAG with Multimodal Document Understanding - arXiv, accessed July 5, 2025, https://arxiv.org/html/2506.16035
38. Advanced Chunking/Retrieving Strategies for Legal Documents : r/Rag - Reddit, accessed July 5, 2025, https://www.reddit.com/r/Rag/comments/1jdi4sg/advanced_chunkingretrieving_strategies_for_legal/
39. SLIDE: Sliding Localized Information for Document Extraction, accessed July 5, 2025, https://arxiv.org/abs/2503.17952
40. RAG for Code Generation: Automate Coding with AI & LLMs - Chitika, accessed July 5, 2025, https://www.chitika.com/rag-for-code-generation/
41. arXiv:2503.09089v2 [cs.SE] 29 Apr 2025, accessed July 5, 2025, https://arxiv.org/pdf/2503.09089
42. Towards Reliable Vector Database Management Systems: A Software Testing Roadmap for 2030 - arXiv, accessed July 5, 2025, https://arxiv.org/html/2502.20812v1
43. HMT: Hierarchical Memory Transformer for Efficient Long Context ..., accessed July 5, 2025, https://arxiv.org/pdf/2405.06067
44. [1602.03218] Learning Efficient Algorithms with Hierarchical Attentive Memory - arXiv, accessed July 5, 2025, https://arxiv.org/abs/1602.03218
45. Daily Papers - Hugging Face, accessed July 5, 2025, https://huggingface.co/papers?q=vector%20database%20retrieval
46. Vector Databases: Tutorial, Best Practices & Examples - Nexla, accessed July 5, 2025, https://nexla.com/ai-infrastructure/vector-databases/
47. What is Vector Embedding? | IBM, accessed July 5, 2025, https://www.ibm.com/think/topics/vector-embedding
48. Vector Databases: Data Storage, Querying, and Embeddings - The ByteDoodle Blog, accessed July 5, 2025, https://blog.bytedoodle.com/vector-databases-data-storage-querying-and-embeddings/
49. When Large Language Models Meet Vector Databases: A Survey - arXiv, accessed July 5, 2025, https://arxiv.org/html/2402.01763v1
50. Hierarchical Repository-Level Code Summarization for Business Applications Using Local LLMs - arXiv, accessed July 5, 2025, https://arxiv.org/html/2501.07857v1
51. Context-aware Code Summary Generation - arXiv, accessed July 5, 2025, https://arxiv.org/html/2408.09006v1

52. Introducing "Files to LLM Prompt" - A VSCode Extension to ... - Reddit, accessed July 5, 2025, https://www.reddit.com/r/PromptEngineering/comments/1imp2xu/introducing_files_to_llm_prompt_a_vscode/
53. Introducing "Files to LLM Prompt" - A VSCode Extension to Streamline Prompting Claude with Your Code : r/ClaudeAI - Reddit, accessed July 5, 2025, https://www.reddit.com/r/ClaudeAI/comments/1ih1tlg/introducing_files_to_llm_prompt_a_vscode/
54. Tips & Tricks for GitHub Copilot Chat in Visual Studio - Learn Microsoft, accessed July 5, 2025, https://learn.microsoft.com/en-us/visualstudio/ide/copilot-chat-context?view=vs-2022
55. DesignCoder: Hierarchy-Aware and Self-Correcting UI Code Generation with Large Language Models - arXiv, accessed July 5, 2025, https://arxiv.org/html/2506.13663
56. Advanced Retrieval for RAG on Code - Reddit, accessed July 5, 2025, https://www.reddit.com/r/Rag/comments/1irf4yh/advanced_retrieval_for_rag_on_code/
57. Context Switching and LLMs – A Cognitive Parallel | David Lozzi, accessed July 5, 2025, https://davidlozzi.com/2025/05/19/context-switching-and-llms-a-cognitive-parallel/
58. Chain of Agents: Large language models collaborating on long-context tasks, accessed July 5, 2025, https://research.google/blog/chain-of-agents-large-language-models-collaborating-on-long-context-tasks/
59. Synergizing RAG and Reasoning: A Systematic Review - arXiv, accessed July 5, 2025, https://arxiv.org/html/2504.15909v1
60. NoLiMa: Long-Context Evaluation Beyond Literal Matching - Finally a good benchmark that shows just how bad LLM performance is at long context. Massive drop at just 32k context for all models. - Reddit, accessed July 5, 2025, https://www.reddit.com/r/LocalLLaMA/comments/1io3hn2/nolima_longcontext_evaluation_beyond_literal/
61. Large Language Models (LLMs) for Source Code Analysis: applications, models and datasets - arXiv, accessed July 5, 2025, https://arxiv.org/html/2503.17502v1
62. CodeTree: Agent-guided Tree Search for Code ... - ACL Anthology, accessed July 5, 2025, https://aclanthology.org/2025.naacl-long.189.pdf
63. [2506.09351] DIVE into MoE: Diversity-Enhanced Reconstruction of Large Language Models from Dense into Mixture-of-Experts - arXiv, accessed July 5, 2025, https://arxiv.org/abs/2506.09351
64. A Comprehensive Survey of Mixture-of-Experts: Algorithms, Theory, and Applications - arXiv, accessed July 5, 2025, https://arxiv.org/html/2503.07137v1
65. MemoryLLM: Towards Self-Updatable Large Language Models - arXiv, accessed July 5, 2025, https://arxiv.org/html/2402.04624v2
66. Dao-AILab/flash-attention: Fast and memory-efficient exact ... - GitHub, accessed

July 5, 2025, https://github.com/Dao-AILab/flash-attention
67. A Novel Round-Level Attention Mechanism to Accelerate LLM Inference - arXiv, accessed July 5, 2025, http://arxiv.org/pdf/2502.15294
68. Here's how I use LLMs to help me write code - Simon Willison's Weblog, accessed July 5, 2025, https://simonwillison.net/2025/Mar/11/using-llms-for-code/
69. GitHub Copilot deep dive: Model selection, prompting techniques & agent mode - YouTube, accessed July 5, 2025, https://www.youtube.com/watch?v=0Oz-WQi51aU&pp=0gcJCfwAo7VqN5tD