BandwidthBuddy: Network Traffic Analysis and Classification using Large Language Models

Samuil Nikolov, Matthew Karten, and Dr. Laxima Kandel

Abstract

BandwidthBuddy is an intelligent network packet analysis system designed to evaluate and classify network packets in real-time using artificial intelligence models. This paper presents a comprehensive exploration of the AI implementation architecture, focusing on the integration of large language models (LLMs) through the Ollama framework. We discuss fundamental concepts in neural networks, model architectures, and training methodologies, with particular emphasis on prompt engineering and fine-tuning strategies. Our research aims to establish a comparative analysis between unmodified large-scale models and domain-specific fine-tuned models to determine optimal performance characteristics in the context of cybersecurity packet evaluation. This paper provides the theoretical foundation and implementation details necessary for understanding the AI-driven approach to network security analysis.

Contents

1 Introduction

Network security analysis represents a critical challenge in modern cybersecurity operations. The exponential growth of network traffic and the increasing sophistication of threats necessitate automated systems capable of rapidly identifying, classifying, and evaluating potentially malicious packets. Traditional rule-based approaches, while effective for known attack signatures, struggle with novel threats and require constant manual updates. The emergence of large language models and general-purpose artificial intelligence systems has opened new possibilities for intelligent packet analysis. These models demonstrate remarkable ability to process complex textual and structured data, recognize patterns, and provide contextual analysis. However, deploying such systems in specialized domains raises important questions about efficiency, accuracy, and resource utilization. BandwidthBuddy addresses these challenges by implementing an AI-powered packet analysis system that leverages language models for threat evaluation and classification. Rather than relying solely on traditional signature-based detection, our approach uses natural language processing capabilities to analyze packet characteristics contextually. The system combines packet capture, real-time visualization, and AI-driven evaluation to provide security analysts with comprehensive threat assessment. The primary research objective of this work is to investigate the performance characteristics of different AI models in packet analysis tasks, with specific focus on comparing large, general-purpose models against smaller, domain-specific fine-tuned variants. The underlying hypothesis is that smaller models, when properly trained on packet analysis data, should demonstrate superior performance in speed, accuracy, and resource efficiency compared to larger unoptimized models. This paper presents the theoretical foundations of modern AI systems, details the implementation architecture of BandwidthBuddy, and outlines the research methodology for model evaluation and comparison.

2 Fundamentals of Artificial Intelligence and Neural Networks

2.1 Core Concepts in Machine Learning

Artificial intelligence broadly encompasses systems capable of performing tasks that typically require human intelligence. Machine learning, a subset of AI, focuses on systems that learn from data rather than following explicitly programmed instructions. The fundamental premise of machine learning is that patterns in data can be extracted and leveraged to make predictions or decisions on unseen data. Supervised learning represents the most common paradigm in modern machine learning applications. In supervised learning, a model learns from labeled training data where each input example is paired with a ground-truth output or label. The model learns to map inputs to outputs by minimizing the difference between predicted and actual outputs, a process formalized through loss functions and optimization algorithms. Neural networks form the computational substrate of modern deep learning systems. A neural network consists of interconnected nodes (neurons) organized in layers. Each connection between neurons carries a weight, a numerical parameter that modulates information flow. The network processes input data through successive layers, applying mathematical transformations at each stage. By adjusting these weights during training, neural networks learn to approximate complex functions mapping inputs to desired outputs.

2.2 Neural Network Architecture and Information Flow

A feedforward neural network processes information through sequential layers: an input layer that receives raw data, one or more hidden layers that perform nonlinear transformations, and an output layer that produces final predictions. During forward propagation, data flows through these layers with each neuron computing a weighted sum of inputs followed by application of a nonlinear activation function. This nonlinearity is essential, as stacking linear transformations would produce only linear functions, severely limiting model expressiveness. The training process relies on backpropagation, an algorithm that computes gradients of the loss function with respect to all network parameters. These gradients indicate the direction and magnitude of parameter adjustments needed to reduce the loss. Optimization algorithms, most commonly variants of stochastic gradient descent, use these gradients to iteratively update parameters, gradually improving model performance on training data. The architecture of neural networks varies considerably based on the problem domain. Convolutional neural networks (CNNs) utilize local connectivity patterns and parameter sharing for efficient processing of spatial data. Recurrent neural networks (RNNs) maintain internal state allowing processing of sequential data. Transformer architectures, the foundation of modern large language models, employ attention mechanisms to establish long-range dependencies and process data in parallel.

2.3 Transformers and Language Models

Transformer architecture, introduced by Vaswani et al., revolutionized natural language processing by enabling efficient parallel processing of sequential data. The core innovation is the attention mechanism, which allows the model to weigh the importance of different input tokens when processing each position. Self-attention enables each token to attend to all previous tokens, capturing long-range dependencies essential for understanding context. Language models based on transformer architecture operate by predicting the next token in a sequence given all previous tokens. During training, these models learn to maximize the probability of the correct next token in massive corpora of text. This autoregressive prediction task, surprisingly, leads to models that develop sophisticated understanding of language, logic, and factual knowledge. After training on diverse text from the internet, these models demonstrate remarkable zero-shot and few-shot learning capabilities across diverse tasks. Large language models (LLMs) refer to transformer-based language models with billions of parameters trained on vast datasets. The scaling laws of language models suggest that both model size and dataset size play crucial roles in determining model capabilities. Models like Gemma, Llama, and Mistral demonstrate that high-quality performance is achievable across a spectrum of model sizes, with careful architecture and training choices enabling smaller models to achieve performance approaching much larger competitors.

3 AI Model Training and Optimization

3.1 Pre-training and Transfer Learning

Modern large language models typically undergo two-stage training: pre-training and fine-tuning. During pre-training, models learn on large diverse datasets without task-specific labels. This stage builds foundational language understanding, factual knowledge, and reasoning capabilities. Pre-training objectives typically involve predicting masked tokens or generating continuations, forcing the model to develop deep understanding of language structure and semantics. Transfer learning leverages knowledge acquired during pre-training. Rather than training specialized models from scratch, practitioners can start with pre-trained weights and adapt them to specific tasks. This approach dramatically reduces training time, data requirements, and computational cost. The pre-trained model has already learned general patterns about language; fine-tuning specializes this knowledge to the target domain. The effectiveness of transfer learning depends on similarity between pre-training and target tasks. Models pre-trained on broad text corpora transfer well to many language tasks because they develop generalizable language understanding. For specialized domains like packet analysis, however, the pre-trained model's knowledge of network security terminology and packet structures may be limited, suggesting potential benefits from domain-specific fine-tuning.

3.2 Context Windows and Sequence Length

Language models process sequences of discrete tokens and operate within fixed-size context windows. A context window represents the maximum number of tokens the model can attend to when generating the next token. Typical context windows range from 2,048 tokens for smaller models to 128,000 or more for larger variants. This constraint fundamentally affects model capabilities and application design. Context window size determines how much historical information influences predictions. Smaller windows force the model to forget older information within a sequence, potentially losing important context. Larger windows enable the model to maintain coherent reasoning over longer documents and conversations, though computational cost increases quadratically with context length due to the attention mechanism's complexity. For application design, context window limitations require careful management. In conversational systems, maintaining conversation history within context window limits necessitates strategies like keeping recent messages while discarding older ones. In packet analysis, where multiple packets may be evaluated in context, context management determines how much surrounding packet information influences individual packet evaluation.

4 Fine-Tuning: Specialization and Domain Adaptation

4.1 Motivation for Fine-Tuning

While pre-trained models demonstrate impressive zero-shot performance, they are not optimized for specific domains or tasks. A general-purpose language model trained on internet text has not specialized in network security terminology, packet structure understanding, or the specific evaluation criteria required for threat assessment. Fine-tuning adapts pre-trained models to these specialized requirements. Fine-tuning involves training the pre-trained model on task-specific labeled data. Unlike pre-training, which requires massive computational resources and enormous datasets, fine-tuning operates on relatively small labeled datasets and requires modest computational overhead. Typically, only a subset of model parameters are updated during fine-tuning, preserving the general knowledge acquired during pre-training while specializing the model's behavior. The central research hypothesis underlying BandwidthBuddy's design is that smaller models fine-tuned specifically for packet analysis should outperform larger general-purpose models when evaluated on packet classification tasks. This hypothesis rests on several observations: (1) smaller fine-tuned models are faster and more resource-efficient, (2) domain-specific training should improve accuracy on task-specific metrics, and (3) the overhead of general capabilities in large models becomes unnecessary burden for narrow specialized tasks.

4.2 Fine-Tuning Methodologies

Full fine-tuning modifies all parameters of the pre-trained model on task-specific data. While effective, this approach consumes significant memory and computation for large models. For practical applications, parameter-efficient fine-tuning techniques have emerged as valuable alternatives. Low-Rank Adaptation (LoRA) represents a significant advance in efficient fine-tuning. Rather than updating all model parameters, LoRA introduces small trainable adapter layers with rank much lower than the original parameters. During fine-tuning, only these low-rank adapters are trained while original model parameters remain frozen. This approach reduces memory requirements and training time while maintaining performance comparable to full fine-tuning. Quantized LoRA (QLoRA) further reduces requirements by operating on quantized model weights. For BandwidthBuddy's development path, LoRA-based fine-tuning emerges as a practical approach. Starting with a small base model like Gemma 3B or Llama 3 8B, domain-specific fine-tuning can be performed on curated packet analysis datasets. The resulting adapter layers provide specialized behavior while keeping the base model unchanged, enabling easy distribution and deployment of the specialized variant.

4.3 Training Data and Annotation

Fine-tuning quality depends fundamentally on training data quality and relevance. For packet analysis, effective training data must include diverse packet examples with accurate labels indicating correct evaluation, severity assessment, and threat categorization. Manual annotation of such datasets requires domain expertise and significant effort. The training process should present examples in the format expected during inference. If packets are presented as structured JSON and the model should output severity scores and threat assessment, training examples should follow identical format and structure. This consistency ensures the fine-tuned model learns the exact behavior required for production deployment. Transfer learning benefits apply within fine-tuning as well. A model fine-tuned on general cybersecurity tasks provides a better initialization for packet analysis fine-tuning than the original pre-trained model. Creating intermediate fine-tuned variants targeting progressively more specific tasks can substantially improve final performance.

5 Prompt Engineering and Model Direction

5.1 Principles of Prompt Engineering

Prompt engineering encompasses techniques for designing inputs that elicit desired outputs from language models. Even before fine-tuning, well-engineered prompts can substantially improve model behavior on specific tasks. Prompt engineering establishes the foundation upon which fine-tuning builds, as the patterns present in effective prompts often become baked into fine-tuned models. Effective prompts follow several principles.

First, role definition establishes the model's intended persona and expertise. Instructing a model to "assume the role of network security analyst" provides critical framing for subsequent instructions. Second, explicit task specification eliminates ambiguity about what the model should do. Vague instructions like "analyze this packet" are inferior to specific instructions like "evaluate only the target packet and provide severity assessment between 0 and 100." Third, format specification constrains outputs to structured, parseable forms. Requiring specific output fields in consistent formats ensures subsequent processing steps can reliably extract information. Fourth, constraint specification prevents undesired behaviors. Explicitly forbidding markdown formatting, requiring numeric outputs, or restricting responses to specific topics guides the model toward desired behavior.

5.2 Structured Prompting for Packet Analysis

BandwidthBuddy's packet evaluation system uses a multi-part prompt structure adapted from best practices. The system prompt, maintained throughout a conversation, defines the analyst role and core constraints. When a user requests packet evaluation, the user prompt provides the packet data and specific evaluation instructions. This two-level structure separates persistent behavioral instructions from variable task inputs. The evaluation prompt specifies required response format with explicit sections for packet identification, severity rating, confidence assessment, and rationale. By requiring structured output, the system ensures evaluation results are machine-parseable and consistent. The prompt explicitly forbids markdown and other formatting that could interfere with downstream parsing. For conversational follow-up questions, a distinct chat system prompt replaces the evaluation prompt while maintaining the security analyst role. This prompt adds topic restrictions, ensuring questions remain focused on the initial packet evaluation and related cybersecurity topics rather than diverging to unrelated subjects. This focused conversation design keeps the model within its intended domain of expertise.

5.3 Prompt Engineering as Fine-Tuning Foundation

Effective prompt engineering directly informs fine-tuning strategy. Patterns successfully established through prompting should be integrated into fine-tuned models as learned behaviors rather than requiring repeated explicit instruction. A model fine-tuned on packet analysis data presented in the same structured format as the prompts will learn to produce those formats automatically. The iterative relationship between prompt engineering and fine-tuning suggests an initial development phase using pure prompt engineering to establish optimal behavior patterns. Once patterns are identified, data collected from this phase becomes training material for fine-tuning. The fine-tuned model should replicate the behavior of the prompted model while requiring shorter, simpler prompts. This development approach allows BandwidthBuddy to begin with existing pre-trained models, establish effective prompt patterns through engineering, then transition to smaller fine-tuned models offering superior efficiency while maintaining performance.

6 System Architecture and Implementation

6.1 Layered System Design

BandwidthBuddy implements a modular architecture separating concerns across distinct layers. The frontend layer provides user interface for packet display, interaction, and chat functionality. The application server layer implements API endpoints, session management, and context window oversight. The Ollama service layer executes model inference on local hardware. The packet sniffer service captures and provides network traffic data. This modular design offers several advantages. Each layer can be developed and tested independently. Service separation enables horizontal scaling and technology flexibility. The abstraction between layers reduces coupling, simplifying modifications to individual components. For example, swapping between different Ollama models requires only configuration changes in the application server, not modifications to frontend or sniffer components.

6.2 Chat Session Management and Context Windows

The chat session manager maintains per-packet conversation history including the system prompt, all previous user messages, and all previous model responses. When evaluating a new packet or responding to follow-up questions, the entire history is sent to Ollama, providing context for coherent conversational responses. Context window management represents a critical implementation detail. As conversation history grows, the total token count may exceed the model's context window. Rather than failing when context limits are reached, the session manager implements trimming logic that maintains the system prompt (never removed) and recent messages while discarding oldest messages. This strategy preserves conversation coherence by maintaining the most recent context where the conversation focus resides. Token estimation guides context management decisions. Using conservative rules-of-thumb (approximately 4 characters per token), the system estimates token counts and implements early trimming before approaching hard limits. For Gemma models with 8,192 token context windows, the system typically reserves the full system prompt plus recent conversation history, with explicit guards ensuring buffer space for model-generated responses.

6.3 Ollama API Integration

Ollama provides a RESTful API for local model execution, standardizing communication between the application server and model runtime. The API accepts requests specifying the model name, message history in ChatML format, and parameters like streaming mode. Requests are synchronous, with responses containing the model's next token in the conversation. The ChatML format standardizes message representation across different models and providers. Each message specifies a role (system, user, or assistant), differentiating behavioral instructions from user inputs from model outputs. This format enables consistent conversation management across different model architectures. The application server constructs requests by gathering the conversation history for a packet, compiling into ChatML format, and posting to Ollama's endpoint. Response parsing extracts the generated text, which is then stored back in the session history. The synchronous request pattern is suitable for the typical packet evaluation workload where analyses complete in seconds.

6.4 Context Aggregation from Packet Sniffer

The packet sniffer service captures network traffic and stores packet data with metadata. When evaluating a target packet, the application server retrieves surrounding packets (configurable count before and after) to provide context. This surrounding context helps the model understand the packet's place in network flow, potentially revealing patterns characteristic of normal traffic versus attacks. The context aggregation process constructs a comprehensive data structure including target packet details and surrounding packets. This structure is formatted into human-readable text describing packet characteristics including source/destination IPs, protocols, payload sizes, flags, and other relevant fields. The formatting prepares data for language model consumption, translating binary network data into structured text.

7 Evaluation Criteria and Comparative Analysis Framework

7.1 Performance Metrics

Evaluation of AI models for packet analysis requires comprehensive metrics addressing multiple performance dimensions. Accuracy measures the fraction of packets correctly classified. For packet analysis, accuracy should be weighted by threat severity, with misclassifying dangerous packets as benign weighted more heavily than misclassifying benign packets as dangerous. Precision and recall formalize the distinction between false positive and false negative errors. Precision measures the fraction of positive classifications that are correct (relevant for reducing false alarms). Recall measures the fraction of actual threats detected (relevant for comprehensive security). The F1 score harmonizes these measures through their harmonic mean. Speed represents critical metric for practical deployment. Evaluation latency directly affects analyst workflow and responsiveness. Storage requirements and memory consumption determine feasibility on resource-constrained systems. These efficiency metrics become particularly important when comparing large models against fine-tuned smaller variants.

7.2 Inference Characteristics

Beyond accuracy, the quality of model reasoning can be assessed through human evaluation of explanations. Even perfectly accurate classifications lack value if analysts cannot understand the reasoning. Models should provide clear rationales for severity assessments, identifying specific packet characteristics supporting conclusions. Consistency measures whether repeated evaluations of identical packets produce identical results. Language models introduce sampling in token generation, potentially producing variable responses even to identical inputs. For security analysis, consistency is essential, as recommendations should not vary arbitrarily between evaluation runs. Calibration of confidence estimates matters for trust and human decision-making. Models should express high confidence for clear threat evaluations while expressing appropriate uncertainty for ambiguous cases. Miscalibrated confidence (high confidence on incorrect answers) erodes analyst trust.

7.3 Comparative Analysis Methodology

Comparative evaluation requires establishing baseline performance from unoptimized general-purpose models on packet analysis tasks. Models like Gemma 3B, Llama 3 8B, and Mistral 7B evaluated with straightforward prompting establish baseline accuracy, speed, and resource usage. These baselines provide reference points for assessing fine-tuned model improvements. Fine-tuned variants of the same base models, trained on packet analysis data with optimized prompts, should demonstrate performance improvements. Comparing fine-tuned models against base models isolates the effect of domain-specific training, controlling for model architecture and size. Comparison across different base model sizes (e.g., fine-tuned 3B versus fine-tuned 8B) reveals efficiency versus performance tradeoffs. The research hypothesis predicts that appropriately fine-tuned smaller models outperform general-purpose larger models on packet analysis metrics. Validating or refuting this hypothesis requires systematic evaluation comparing accuracy, speed, and resource usage across multiple models and training configurations.

8 Human Factors and User Interface Considerations

8.1 Usability in Security Analysis Workflow

While AI model capabilities form the technical foundation of BandwidthBuddy, practical effectiveness depends on successful integration into security analysts' workflows. User interface design significantly impacts whether the system's capabilities translate to improved analyst productivity and threat detection. Effective packet visualization presents relevant information with appropriate visual hierarchy. Critical indicators like threat severity should dominate visual space. Supporting details like protocol information, IP addresses, and payload size should be accessible but not overwhelming. Interactive filtering and search enable analysts to quickly navigate large packet sets. The chat interface bridges gap between raw AI output and analyst understanding. Well-designed conversation features enable natural follow-up questions, clarification of model reasoning, and collaborative exploration of packets. The interface should clearly distinguish model-generated content from actual packet data, preventing analysts from mistaking AI hallucinations for facts about network traffic. Responsiveness significantly affects user experience. Long evaluation latencies frustrate users and break workflow continuity. Streaming model outputs can provide partial results immediately rather than requiring full completion before displaying anything, improving perceived responsiveness even when total latency remains unchanged.

8.2 Visualization Architecture (Pending Implementation)

The packet visualization subsystem represents the visual and interactive component of the user interface. Future development will establish how packet data flows from capture through visualization, how interactive elements enable exploration, and how AI evaluations integrate with visual presentation. Pending design considerations include determining appropriate packet representation (hierarchical tree, tabular, timeline-based), establishing visual encoding schemes for packet properties, and designing interactive selection mechanisms. The visualization architecture should accommodate both high-level overview of many packets and deep drill-down into individual packet details.

8.3 Human-in-the-Loop Analysis

BandwidthBuddy's design emphasizes human-in-the-loop analysis where AI serves as analyst assistant rather than autonomous decision-maker. The system should present AI evaluations as recommendations warranting human verification rather than definitive judgments. This framing preserves analyst agency and acknowledges that context unknown to the system may warrant departing from AI recommendations. Features supporting human oversight include confidence indicators on AI assessments, explanations of reasoning, and easy mechanisms for recording analyst disagreement with AI evaluation. This feedback loop can inform both immediate analyst decisions and longer-term model retraining.

9 Implementation Status and Development Roadmap

9.1 Current Implementation

BandwidthBuddy currently implements the core AI integration architecture using Gemma 3B as the baseline model, accessed through Ollama. The application server in Node.js manages chat sessions, handles context windows, and coordinates with both the Ollama service and packet sniffer. The frontend provides basic interface for packet selection and AI evaluation results. System prompts establish the security analyst role and structured output format. Evaluation prompts present target and context packets in formatted text, with explicit instructions to assess severity and confidence. Chat prompts enable follow-up questions constrained to cybersecurity topics. Initial implementation validates the technical architecture and establishes baseline performance metrics for later comparison with fine-tuned variants. Current focus is on ensuring reliable integration between system components and consistent, parseable model output.

9.2 Prompt Engineering Phase

Current development emphasizes prompt engineering to establish optimal behavior patterns. Iteration on prompt structure, format specifications, and constraint definitions aims to maximize accuracy and consistency with available models. This phase generates performance baseline and collects patterns successful in eliciting desired model behavior. Data collected during this phase becomes training material for subsequent fine-tuning. Evaluating diverse packet examples with various prompts builds understanding of which prompt patterns yield reliable, accurate outputs across different packet types and threat levels.

9.3 Planned Fine-Tuning Research

Planned fine-tuning research will train smaller models on packet evaluation data accumulated during prompt engineering phase. Training should follow LoRA-based approaches to maintain efficiency. Multiple model sizes and architectures will be trained to establish performance curves showing tradeoffs between model size and accuracy. Comparative evaluation will assess whether fine-tuned small models outperform unoptimized large models on target metrics. Results will validate or refute the core research hypothesis regarding efficiency of domain-specific small models.

9.4 Packet Sniffer and Visualization Development

The packet sniffer service requires development to capture network traffic reliably across different network interfaces and operating systems. The sniffer must efficiently store packet data and support flexible queries for retrieving packets by ID and surrounding context. Visualization system development will establish user interface design translating network packet data into intuitive visual representation. Integration with AI evaluations will present threat assessments alongside packet visualization, enabling analysts to correlate visual evidence with AI recommendations. These components represent critical system functionality but are outside the immediate scope of the AI model investigation presented in this paper.

10 Challenges and Considerations

10.1 Accuracy and Reliability

Language model outputs can reflect training data biases and can hallucinate plausible-sounding but factually incorrect information. For security applications where errors have real consequences, this limitation is serious. Fine-tuning on accurate labeled data can partially mitigate hallucination, but cannot entirely eliminate it. Validating AI recommendations against human expert assessment remains essential. A model should augment analyst capability rather than replace human judgment. Systems design should preserve analyst override and disagreement mechanisms.

10.2 Privacy and Data Handling

Packet analysis inherently involves network traffic potentially containing sensitive information. Handling this data responsibly requires careful consideration of data storage, access controls, and inference processing. Local execution through Ollama avoids transmitting packet data to external services, reducing privacy exposure compared to cloud-based alternatives.

10.3 Computational Efficiency

Larger models consume substantial computational resources. For practical deployment on analyst workstations or security monitoring infrastructure, resource efficiency becomes important. Smaller fine-tuned models address this limitation by providing comparable performance with lower computational overhead. Context window management affects inference cost. Longer conversations consume more tokens, increasing computational cost and latency. Strategic context trimming balances conversation coherence against efficiency.

11 Conclusion

BandwidthBuddy represents an AI-driven approach to network packet analysis addressing the growing need for intelligent threat detection in complex modern networks. By integrating language models through the Ollama framework, the system leverages sophisticated AI capabilities while maintaining practical feasibility through local execution and efficient inference. This paper has established the theoretical foundations underlying the implementation: how neural networks learn, how transformers enable language understanding, how fine-tuning adapts general models to specialized tasks, and how prompt engineering guides model behavior toward desired outputs. These foundations directly inform BandwidthBuddy's design and planned research comparing large general-purpose models against small fine-tuned specialists. The core research hypothesis—that properly fine-tuned smaller models outperform larger unoptimized models on packet analysis tasks—remains to be validated through systematic comparative evaluation. Current implementation establishes technical feasibility and begins the prompt engineering phase that will generate training data for future fine-tuning efforts. Future development will implement comprehensive evaluation infrastructure, conduct fine-tuning research, and integrate visualization and packet sniffing components. The resulting system should demonstrate that domain-specific AI adaptation provides practical advantage in security applications while maintaining computational efficiency. By grounding the system in sound AI principles while maintaining focus on practical security analysis requirements, BandwidthBuddy aims to advance both the state of AI-powered security tools and understanding of model optimization for specialized domains.

References

References

[1] Vaswani, A., et al. (2017). “Attention Is All You Need.” *Advances in Neural Information Processing Systems.*

[2] Devlin, J., et al. (2018). “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.” *arXiv preprint arXiv:1810.04805.*

[3] Radford, A., et al. (2019). “Language Models are Unsupervised Multitask Learners.”

[4] Brown, T. A., et al. (2020). “Language Models are Few-Shot Learners.” *Advances in Neural Information Processing Systems, 33*, 1877–1901.

[5] Hu, E. Y., et al. (2021). “LoRA: Low-Rank Adaptation of Large Language Models.” *arXiv preprint arXiv:2106.09685.*

[6] Wei, J., et al. (2021). “Finetuned Language Models are Zero-Shot Learners.” *International Conference on Learning Representations.*

[7] Kaplan, J., et al. (2020). “Scaling Laws for Neural Language Models.” *arXiv preprint arXiv:2001.08361.*

[8] Hoffmann, J., et al. (2022). “Training Compute-Optimal Large Language Models.” *arXiv preprint arXiv:2203.15556.*

Appendix: Architecture Diagrams

Figure A1: System Architecture Overview

`placeholder_system_architecture.png`

Figure 1: High-level system architecture showing data flow between frontend interface, application server, Ollama service, and packet sniffer. The frontend communicates with the backend through HTTP/WebSocket, the backend manages sessions and coordinates with Ollama for model inference and with the packet sniffer for context data.

Figure A2: Ollama API Request-Response Flow

`placeholder_ollama_flow.png`

Figure 2: Detailed request-response flow between application server and Ollama service. The server constructs a request containing the model name and message history in ChatML format, Ollama processes the request through model inference, and returns the generated response which is integrated back into session history.

Figure A3: Chat Session State Management

placeholder_session_management.png

Figure 3: Session state evolution from initial packet evaluation through conversational follow-up. Sessions begin with system prompt initialization, transition through packet evaluation with structured output, and continue into chat mode where conversational follow-up questions maintain context while system prompt is replaced with chat-specific instructions.

Figure A4: Context Window Management Strategy

placeholder_context_window.png

Figure 4: Context window allocation strategy showing system prompt protection, reserve space for model output, and available space for conversation history. Trimming operations maintain system prompt while discarding oldest messages when conversations approach context limits, preserving recent context critical for coherent responses.