Trusted LLM Inference on the Edge with Smart Contracts

Rabimba Karanjai
Department Of Computer Science
University Of Houston
rkaranjai@uh.edu

Weidong Shi
Department Of Computer Science
University Of Houston
wshi3@uh.edu

2024 IEEE International Conference on Blockchain and Cryptocurrency (ICBC) | 979-8-3503-1674-2/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICBC59979.2024.10634448

***Abstract*—In this era, significant transformations in industries and tool utilization are driven by AI/Large Language Models (LLMs) and advancements in Machine Learning. There's a growing emphasis on MLOps for managing and deploying these AI models, along with a focus on distributed inferences. Concurrently, the imperative for secure on-chain computation is escalating. Our paper introduces an innovative framework that integrates blockchain technology, particularly the Cosmos SDK, to facilitate distributed AI inferences on edge devices. This system, built on WebAssembly (WASM), enables interchain communication and deployment of WASM modules executing AI inferences across multiple blockchain nodes. We critically assess this system's safety, scalability, and model security, with a special focus on its portability and engine-model agnostic deployment on edge devices.**

I. Introduction

The field of Artificial Intelligence (AI) has witnessed significant progress, particularly in Large Language Models (LLMs) like Generative Pre-trained Transformers (GPTs) [1] and Google PaLM 2 [2]. These advancements have led to the integration of LLMs into practical applications, as evidenced by research in [3], [1]. One particularly captivating application lies in the generation of source code from natural language instructions. This development, exemplified by systems like ChatGPT and Github CoPilot [4], has the potential to revolutionize the programming process by enabling programmers to express their intent in natural language and have it automatically translated into functional code across various programming paradigms.

In computer science, the exploration and integration of decentralized ledgers and smart contracts have marked a pivotal shift within the financial sector. This shift is highlighted by the extensive use of Uniswap's smart contracts, which supported transactions surpassing $7.17 billion in 2021 [5]. The utility of smart contracts extends across various domains, including but not limited to, Confidential Computing [6], [7], Decentralized Serverless Functions [8], and Event-based Transactions [9], [10]. Given this backdrop, an essential inquiry arises concerning the feasibility and efficacy of employing artificial intelligence (AI) and large language models (LLMs) within a distributed and secure smart contract environment for the purpose of financial modeling and forecasting. This inquiry seeks to determine whether such integration could enhance the efficiency and functionality of the financial workflow.

In this research, we delve into the potential of executing smart contracts on a blockchain to determine if they can trigger AI agent operations based on those contracts. Our investigation focuses on whether a system can be designed to seamlessly integrate with the existing smart contract workflow, avoiding disruptive changes and adhering to current standards. This would enable the application of AI-driven finance models to produce outputs that are numerical, textual, and multimodal in nature. Our primary interest lies in whether our envisioned system could integrate within established frameworks, offering an innovative method for interacting with AI models, including large language models (LLMs), in a manner that is conscious of privacy concerns.

We introduce a conceptual framework aimed at facilitating decentralized, on-device edge inferences that are agnostic to the specific AI models, through smart contracts. These contracts act as triggers for the AI inference engine, which operates within a rust based wasm runtim runtime environment. Our research is directed at addressing several key questions:

RQ1: Can smart contracts effectively incorporate AI and LLM inferences while maintaining reasonable performance on edeg devices? We explore the possibility of developing a workflow on existing smart contract platforms that is indifferent to the type of AI model or LLM used.

RQ2: Is it feasible to perform inference locally in low powered and often disconnected edgde devices in a manner that is aware of privacy considerations? We examine the potential for smart contracts to directly invoke models on-device, thereby retrieving inferences in a secure manner.

RQ3: Are these secure The security implications of implementing such a framework, including its ad-

vantages and potential drawbacks, are scrutinized. We demonstrate how our framework can inherently protect against certain types of covert channel attacks that could occur during native model inference.

To tackle these questions, we introduce the concept of Trusted LLM Inference on the Edge with Smart Contracts, or TLIESC, building on [11] for blockchain execution of smart contracts and incorporating our reference implementation for AI inferences with open-source models.

In conclusion, our contributions through this study include:

- The proposition of TLIESC as a method for integrating AI and LLM execution and response within the context of smart contract execution.
- The demonstration of the feasibility for executing AI and LLM inferences on edge through smart contracts using local models.
- Evidence that our framework can be AI model and inference engine agnostic, offering adaptability for future systems.

II. Related Work

Recent AI advancements offer new opportunities for blockchain technologies, particularly through large language models (LLMs) and generative AI, which are seen as transformative for financial analysis and decision-making [12], [13]. Cosmos blockchains enable smart contract execution via WASM, facilitating high-performance computing and AI applications in smart contracts through its modular SDK and WASM module integration with other Cosmos modules for enhanced functionalities.

Transformer models, specifically the GPT series, utilize the Transformer architecture for predictive language tasks by iterating over token probabilities, a process distinct from fixed-time models like ResNet [1], [14], [15]. TensorFlow Serving and Triton inference serving systems support DNNs, optimizing GPU usage for LLMs, despite memory constraints. Techniques like caching and iteration-level scheduling by Fairseq, HuggingFace, FasterTransformer, and Orca improve computational efficiency and manage GPU memory for interactive applications [16], [17], [18], [19], [20], [21].

Blockchain serves as a decentralized data management system, crucial for applications such as digital currencies and DeFi, through a peer-to-peer network that ensures transaction integrity without central authorities [22], [23], [24], [25]. Its infrastructure supports a dual-layer system of a physical network and blockchain consensus mechanisms, making it foundational for enhancing AI's reliability and trustworthiness.

Sipola et al. [26] review the progress in Edge AI, highlighting both hardware and software developments, with an emphasis on neural network optimization and tools for mobile and microcontroller applications. Imran et al. [27] explore development boards for Edge AI, detailing hardware options for AI algorithms at the edge. Merenda et al. [28] discuss machine learning models, architectures, and implementation requirements for IoT devices, notably on microcontrollers for number detection, emphasizing the deployment on resource-constrained devices. These reviews enhance understanding of edge computing devices' capabilities and considerations for deploying deep learning algorithms at the edge, relevant to our edge-device analysis research.

III. Technical Implementation Details

This study employs the Cosmos Network [11] as its underlying blockchain infrastructure for smart contract execution. The Cosmos ecosystem treats smart contracts as programmable entities that enable autonomous communication and transactions between disparate blockchains. These contracts enforce predefined rules without the need for ongoing human intervention, ensuring the network's decentralized operation.

To support the development and deployment of these smart contracts, the Cosmos platform offers two key tools:

- CosmWASM [29]: A flexible smart contract development framework that allows the creation of custom smart contracts tailored to specific use cases.
- Ethermint [30]: A compatibility layer enabling the execution of Ethereum-based smart contracts within the Cosmos environment, fostering interoperability and leveraging existing Ethereum code.

For this research, we utilize CosmWASM's native smart contract support to deploy and execute the smart contracts relevant to our investigation.

A. Edge Agent

We use CosmWASM as Edge Agent for our platform. CosmWASM emerges as a bespoke framework designed for the creation of smart contracts within the Cosmos blockchain ecosystem, employing WebAssembly (WASM) as its underlying runtime environment. This configuration enables the crafting of high-performance, secure contracts, with a notable advantage being its language-agnostic approach.

On the other hand, Ethermint serves as a conduit linking the Cosmos and Ethereum blockchain ecosystems, functioning essentially as a compatibility layer. This "translator" facilitates the integration of existing

Ethereum-based contracts into the Cosmos infrastructure without necessitating comprehensive redevelopment. By unifying Cosmos's high-speed transaction processing capabilities with the modular architecture of the Cosmos SDK, Ethermint affords developers a rich palette of components for creative contract development.

B. rust-WASM

WebAssembly (WASM), established by the World Wide Web Consortium (W3C) in 2015, represents an endeavor to develop a uniform, high-performance, and secure machine-independent bytecode standard. It prioritizes safety by segregating memory into distinct areas: the stack, global variables, and a linear memory space, all of which are accessed through type-safe operations.

In the context of edge computing, the architectural principles of WASM play a pivotal role in enabling rapid and secure data processing. By not using features that pose security risks while maintaining compatibility with C/C++, WASM effectively addresses a host of technological hurdles.

C. WASMedge

WASMEdge emerges as a pivotal tool for implementing WebAssembly[31] (WASM) within edge computing environments, enhancing the deployment of serverless functions across diverse software infrastructures. Its utility spans a wide range of applications, from facilitating operations at the periphery of cloud networks to serving as an API endpoint within a Function as a Service (FaaS) framework, operating through Node command line interfaces, and extending its functionality to embedded systems [31].

D. WebAssembly Binary

The WebAssembly System Interface (WASI)[32] serves as a standardized bridge between WASM applications and the operating system, mirroring POSIX for cross-platform compatibility and facilitating access to OS services. WASI's capability-based security model enhances application security by confining them to specified resources.

For our framework we use WASMEdge since it is the fastest WASM VM available at the moment [33], [31]. We explain the high level architecture of TLIESC as shown in Fig. 1.

WASMEdge is chosen for its speed among WASM virtual machines, according to benchmarks[33], [31]. We detail our framework's architecture and demonstrate its application through a modified sample smart contract from the Cosmos SDK[34], which dynamically generates input prompts for the LLM inference engine, facilitated by WASMEdge. This process culminates in the execution of smart contracts, seamlessly integrating with the LLM inference engine to produce desired outcomes.

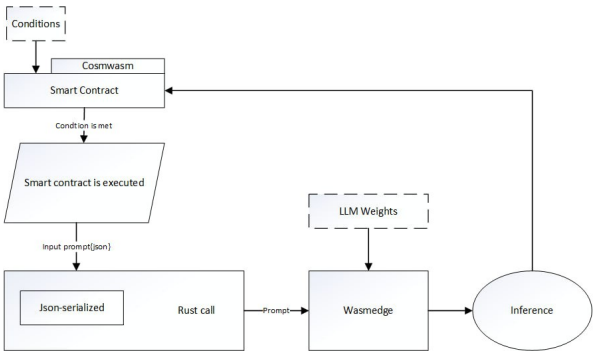

Fig. 1. Overview of TLIESC.

IV. Implementation Highlights

For TLIESC we decided to use a fairly small but capable model, Google's recently released gemma-2b [35] for our base model. This decision was made for the fairly advanmced capaibility of the model even though being a smaller variant of the LLMs released. Which illustrates the usecase better with actually going for a real world model, and not a toy example. Before however we can pass the inputs to the model, CosmWasm modules has to compile and run the contract.

The compilation process is same for both the smart contract and the rust inference engine we wrote for loading the model.

A. Compiling the Contract

The contract written in Rust has to be compiled to wasm code to be executed. The easiest way for us to do that was to use "cargo". We use the following to optimize and compile it to a wasm runtime

Listing 1. WasmCosm Contract Compile

```
RUSTFLAGS='-C link-arg=-s' cargo
    wasm
```

Once we have the compiled wasm file, this was deployed in the blockhain.

Listing 2. WasmCosm Contract Store

```
wasmd tx wasm store nameservice.wasm
    --from <your-key> --chain-id <
    chain-id> --gas auto
```

B. Gemma 2 Inference

For the development of our inference engine, we selected WasmEdge to leverage the WebAssembly (WASM) runtime environment. This decision necessitated a preliminary step: the creation of a program

capable of executing large language model (LLM) inferences, both as a proof of concept and to evaluate feasibility. Our approach draws inspiration from the distinguished gemma.cpp project[36], leading to the development of a streamlined Rust program dedicated to conducting the inferences. The choice of Rust as the programming language was strategic, aimed at minimizing compatibility issues between our smart contract execution framework on CosmWASM and the inference engine. This is due to Rust's capability to be seamlessly integrated within the smart contract environment, thereby obviating the need for any external messaging protocols for its operation.

Listing 3. Wasmedge Code

```
wasmedge --dir .:. --nn-preload
    default:GGML:AUTO:gemma-2b-it-
    Q5_K_M.gguf llama-api-server.wasm
    -p gemma-instruct -c 4096

[USER]:
Who is the father of computer
ASSISTANT
The father of computer is Charles
    Babbage an English mathematician
    philosopher and inventor Babbage
    is considered one of the pioneers
     of computer science and his work
     laid the foundation for modern
    computers
[USER]:
Was he the only one?
[ASSISTANT]:
Sure Babbage was not the only one
    Other pioneers of computer
    science include

Ada Lovelace English mathematician
    and scientist considered one of
    the first programmers
John von Neumann American
    mathematician and computer
    scientist inventor of the
    storedprogram computer
These are just a few of the many
    pioneers of computer science
    Babbages work was a major
    contribution and his
    contributions are still felt
    today
```

V. Portability

One of the key benefits to our proposed solution is model portability and inference engine (as well as model) agnostic nature. Even though we had chosen WasmEdge to showcase our proposed framework, this can easily be replaced by any other code that can communicate directly with a Web Assembly runtime. The primary reason we chose WasmEdge or any Web Assembly runtime because of their ability to communicate with the WebGPU [37] API. We have also tested our implementation using the well documented WGPU [38] API.

A. Inference Engine Portability

Rust is used to construct the demo inference code, which is compiled to WebAssembly. This core Rust code is only forty lines long and is surprisingly small. It can process inputs from the user, log the flow of the conversation, modify the text to make it compatible with the llama2 input template, and do inference tasks using the WASI NN API. This simplified method demonstrates how well the code manages these operations and how efficient it is.

Listing 4. Rust Inference

```rust
fn main() {
    let arguments: Vec<String> = env::
    args().collect();
    let model_identifier: &str = &
    arguments[1];

    let neural_network =
        wasi_nn::GraphBuilder::new(
    wasi_nn::GraphEncoding::Ggml, wasi_nn
    ::ExecutionTarget::AUTO)
            .assemble_from_cache(
    model_identifier)
            .expect("Successful␣build");
    let mut execution_context =
    neural_network.
    create_execution_context().expect("
    Context␣initialization");

    let assistant_directive = String::
    from("<<SYS>>You␣are␣a␣helpful,␣
    respectful␣and␣honest␣assistant.␣
    Answer␣briefly␣and␣safely.␣<</SYS>>")
    ;
    let mut historical_prompt = String::
    new();

.................
.................
.................

        // Output retrieval.
        let mut result_buffer = vec![0u8;
     1000];
        let result_size =
    execution_context.obtain_output(0, &
```

```
    mut result_buffer).expect("Output␣
    retrieval");
21      let response = String::
    from_utf8_lossy(&result_buffer[..
    result_size]).to_string();
22      println!("Response:\n{}",
    response.trim());
23
24      historical_prompt.push_str(&
    format!("{}␣{}", combined_prompt,
    response.trim()));
25  }
26 }
```

This can be compiled using

Listing 5. Rust Compile

```
curl --proto '=https' --tlsv1.2 -sSf
    https://sh.rustup.rs | sh
rustup target add wasm32-wasi
```

Our handwritten Rust code is also definitely not a requirement. To showcase portability we had directly taken the llama2.c [39] engine and compiled it into wasm file.

Listing 6. llama2.c wasm

```
$CC run.c -D_WASI_EMULATED_MMAN -
    lwasi-emulated-mman -
    D_WASI_EMULATED_PROCESS_CLOCKS -
    lwasi-emulated-process-clocks -o
    run.wasm
```

And that also works fine.

B. Model Portability

Even though we have primarily targeted the Llama 2 family of models. We are not bounded by them. Since we base our implementation on the llama.cpp [40], the model formats have to be ggml. For which we utilize the GGML plugin for WasmEdge [41]. We also have tested the framework with LLAVA [42] which is a multi-modal model. However our inference code is not equipped to handle non-text inputs, hence we only tested the text input and output of the model. However, a base64 encoder and decoder is certainly possible to pass multi-modal (image, numbers, text) input to the model to get its feedback.

TLIESC works as shown in Fig 2.

The validator node running in Cosmwasm generates the input message to be passed to the x/aiwasm runtime (which in this case is WasmEdge) which runs the inferences.

WasmEdge can utilize both GPU or CPU based on what is available in the host system. It uses WebGPU API to see if there is a compatible GPU available, and runs the inference if it is available. If not then tries to run the inference in CPU. TLIESC is compatible with other Cosmos modules since it does not modify anything within the CosmWasm, but only individual contract.

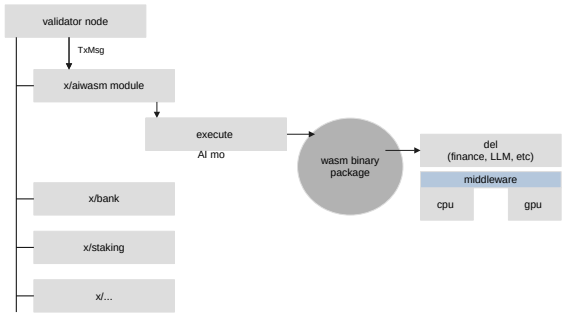

Fig. 2. Working of TLIESC.

Our assumption for this work is the edge devices might or might not have GPU access, hence the chosen gemma model and the gemma.cpp works so well, since it is highly otimized for running using only CPU as well, which was also our tesbed.

VI. Discussion

Our evaluation of LLM/AI execution alongside CosmWASM was conducted on multiple smaller VM's deliberately created with very low device configuration to simulate edge devices running on a host featuring a Intel I9 Processor, NVIDIA 4080 Mobile graphics processor, and 16GB of RAM. This hardware selection, deliberately moderate in configuration, was aimed at identifying any system limitations. During our tests, both models executed within a reasonable timeframe, with the gemma model achieving 28 tokens per second.

A. CosmWASM and Wasm Contract

We adapted the namespace example contract from the Cosmos GitHub repository to include our Rust code. This modification allowed us to successfully execute the compiled WASM smart contract, as outlined in Section IV-A.

RQ1 *Can smart contracts utilize LLM and AI inferences with reasonable performance?*
Our proof of concept demonstrated the feasibility of executing a LLM/AI inference engine on moderately configured hardware to produce model outputs. Within our scope, smart contracts, specifically those executed using CosmWASM, were capable of generating LLM inferences through our framework, highlighting that LLMs are computationally more intensive than other AI models.

B. Data Privacy

Given that inputs are executed directly from the contract and compiled into a WASM file, data security is contingent on the host machine's security. Additionally, since inference runs on the host or a networked machine, model and inference data remain secure as long as system integrity is maintained.

> **RQ2** *Can we get inference locally in a privacy-aware way?*
> Inferences are conducted locally, with compiled models also stored locally, ensuring privacy.

C. Model Performance

Our utilization of WasmEdge for model evaluation allowed full utilization of the WebGPU API. This access provides flexibility in using the host system's GPU for inference tasks or reverting to CPU execution depending on the inference engine's requirements.

> **RQ3** *What are the security implications?*
> Our framework demonstrates mitigation against attacks like *LeftoverLocals*, but continuous effort is required to address future security threats.

D. Potential Applications

Integrating an AI inference engine within smart contract execution opens up new avenues in DeFi, decentralized insurance, DAO governance, and other areas, presenting novel application possibilities.

E. Remarks

The necessity of gas fees for AI inferences in blockchain types (e.g., permissioned vs. permissionless) and the potential adaptation of this concept to the Ethereum Virtual Machine (EVM) highlight areas for further exploration. Additionally, the application of AI inferences in side-chains or layer 2 networks presents an interesting direction for future research.

VII. Future Work

A. WebGPU Native

With WebGPU receiving widespread support, it offers significant advantages over WebGL in efficiency and features such as storage textures and compute shaders. This enhancement is critical for our use case, providing substantial performance improvements. Current methods for running AI inferences with WebGPU include hand-coded models and WASM-backed solutions, pointing towards a need for a more scalable and efficient framework.

B. Usage of WebAssembly (WASM)

Our framework, employing LLM and AI inferences in Rust for CPU and GPU compatibility, addresses web computation limitations. WebAssembly's reduction of JavaScript overhead and the potential for a single WASM file compilation from both Rust and C++ simplify programming logic. Future work will explore a WebGPU framework for language-agnostic, message-passed inferences and a method for streaming model weights to enhance smart contract integration for AI agent construction and blockchain-based applications.

VIII. Conclusion

In conclusion, our research addresses vital questions regarding the integration of AI and Large Language Models (LLMs) with smart contracts. We demonstrated through a proof of concept that smart contracts, particularly those executed using CosmWasm, can effectively generate AI/LLM inferences on edge devices. The use of lightweight models along with CPU inference shows how smart contracts can be used ofr conditional inferencing in a distriburted way. This opens up new avenues for blockchain applications, leveraging the power of AI and LLMs while maintaining security standards.

References

[1] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[2] R. Anil, A. M. Dai, O. Firat, M. Johnson, D. Lepikhin, A. Passos, S. Shakeri, E. Taropa, P. Bailey, Z. Chen *et al.*, "Palm 2 technical report," *arXiv preprint arXiv:2305.10403*, 2023.

[3] J. Jang, S. Kim, S. Ye, D. Kim, L. Logeswaran, M. Lee, K. Lee, and M. Seo, "Exploring the benefits of training expert language models over instruction tuning," 2023. [Online]. Available: https://arxiv.org/abs/2302.03202

[4] J. Finnie-Ansley, P. Denny, B. A. Becker, A. Luxton-Reilly, and J. Prather, "The robots are coming: Exploring the implications of openai codex on introductory programming," in *Proceedings of the 24th Australasian Computing Education Conference*, ser. ACE '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 10–19. [Online]. Available: https://doi.org/10.1145/3511861.3511863

[5] J. Benson, "Uniswap trading volume exploded by 450% to $7 billion. here's why," 2021. [Online]. Available: https://decrypt.co/63280/uniswap-trading-volume-exploded-7-billion-heres-why

[6] K. Rabimba, L. Xu, L. Chen, F. Zhang, Z. Gao, and W. Shi, "Lessons learned from blockchain applications of trusted execution environments and implications for future research," in *Workshop on Hardware and Architectural Support for Security and Privacy*, ser. HASP '21. New York, NY, USA: Association for Computing Machinery, 2022. [Online]. Available: https://doi.org/10.1145/3505253.3505259

[7] R. Karanjai, Z. Gao, L. Chen, X. Fan, T. Suh, W. Shi, and L. Xu, "Dhtee: Decentralized infrastructure for heterogeneous tees," in *2023 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2023, pp. 1–3.

[8] R. Karanjai, L. Xu, N. Diallo, L. Chen, and W. Shi, "Defaas: Decentralized function-as-a-service for emerging dapps and web3," in *2023 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2023, pp. 1–3.

[9] R. Karanjai, L. Xu, Z. Gao, L. Chen, M. Kaleem, and W. Shi, "Privacy preserving event based transaction system in a decentralized environment," in *Proceedings of the 22nd International Middleware Conference*, ser. Middleware '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 286–297. [Online]. Available: https://doi.org/10.1145/3464298.3493401

[10] ——, "On conditional cryptocurrency with privacy," in *2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2021, pp. 1–3.

[11] J. Kwon and E. Buchman, "Cosmos whitepaper," *A Netw. Distrib. Ledgers*, vol. 27, 2019.

[12] L. Cao, "Ai in finance: A review," 2020. [Online]. Available: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3647625

[13] D. Krause, "Large language models and generative ai in finance: An analysis of chatgpt, bard, and bing ai," 2023. [Online]. Available: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4511540

[14] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[15] A. Gujarati, R. Karimi, S. Alzayat, W. Hao, A. Kaufmann, Y. Vigfusson, and J. Mace, "Serving {DNNs} like clockwork: Performance predictability from the bottom up," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, 2020, pp. 443–462.

[16] C. Olston, N. Fiedel, K. Gorovoy, J. Harmsen, L. Lao, F. Li, V. Rajashekhar, S. Ramesh, and J. Soyke, "Tensorflow-serving: Flexible, high-performance ml serving," *arXiv preprint arXiv:1712.06139*, 2017.

[17] (2024) The triton inference server. [Online]. Available: https://github.com/triton-inference-server/server

[18] M. Ott, S. Edunov, A. Baevski, A. Fan, S. Gross, N. Ng, D. Grangier, and M. Auli, "fairseq: A fast, extensible toolkit for sequence modeling," *arXiv preprint arXiv:1904.01038*, 2019.

[19] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz *et al.*, "Huggingface's transformers: State-of-the-art natural language processing," *arXiv preprint arXiv:1910.03771*, 2019.

[20] C. Chelba, M. Chen, A. Bapna, and N. Shazeer, "Faster transformer decoding: N-gram masked self-attention," *arXiv preprint arXiv:2001.04589*, 2020.

[21] G.-I. Yu, J. S. Jeong, G.-W. Kim, S. Kim, and B.-G. Chun, "Orca: A distributed serving system for {Transformer-Based} generative models," in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, 2022, pp. 521–538.

[22] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized business review*, 2008.

[23] H. Luo, Y. Wu, G. Sun, H. Yu, S. Xu, and M. Guizani, "Escm: An efficient and secure communication mechanism for uav networks," *arXiv preprint arXiv:2304.13244*, 2023.

[24] G. Zyskind, O. Nathan *et al.*, "Decentralizing privacy: Using blockchain to protect personal data," in *2015 IEEE security and privacy workshops*. IEEE, 2015, pp. 180–184.

[25] B. Cao, Z. Wang, L. Zhang, D. Feng, M. Peng, L. Zhang, and Z. Han, "Blockchain systems, technologies and applications: A methodology perspective," *IEEE Communications Surveys & Tutorials*, 2022.

[26] T. Sipola, J. Alatalo, T. Kokkonen, and M. Rantonen, "Artificial intelligence in the iot era: A review of edge ai hardware and software," in *2022 31st Conference of Open Innovations Association (FRUCT)*. IEEE, 2022, pp. 320–331.

[27] H. A. Imran, U. Mujahid, S. Wazir, U. Latif, and K. Mehmood, "Embedded development boards for edge-ai: A comprehensive report," *arXiv preprint arXiv:2009.00803*, 2020.

[28] M. Merenda, C. Porcaro, and D. Iero, "Edge machine learning for ai-enabled iot devices: A review," *Sensors*, vol. 20, no. 9, p. 2533, 2020.

[29] (2024) Cosmwasm: Basic cosmos-sdk app with web assembly smart contracts. [Online]. Available: https://github.com/CosmWasm/wasmd

[30] (2024) Ethermint. [Online]. Available: https://docs.ethermint.zone/

[31] (2024) Wasmedge. [Online]. Available: https://webassembly.org/

[32] L. Clark, "Standardizing wasi: A system interface to run webassembly outside the web," *Mozilla Hacks–the Web developer blog*, 2019.

[33] S. Zheng, H. Wang, L. Wu, G. Huang, and X. Liu, "Vm matters: a comparison of wasm vms and evms in the performance of blockchain smart contracts," *arXiv preprint arXiv:2012.01032*, 2020.

[34] (2024) cw-contracts. [Online]. Available: https://github.com/deus-labs/cw-contracts/blob/main/contracts/nameservice/src/msg.rs

[35] G. Team, R. Anil, S. Borgeaud, Y. Wu, J.-B. Alayrac, J. Yu, R. Soricut, J. Schalkwyk, A. M. Dai, A. Hauth *et al.*, "Gemini: a family of highly capable multimodal models," *arXiv preprint arXiv:2312.11805*, 2023.

[36] (2024) google/gemma.cpp: lightweight, standalone c++ inference engine for google's gemma models. [Online]. Available: https://github.com/google/gemma.cpp

[37] B. Kenwright, "Introduction to the webgpu api," in *ACM SIGGRAPH 2022 Courses*, 2022, pp. 1–184.

[38] (2024) wgpu: Cross-platform, safe, pure-rust graphics api. [Online]. Available: https://github.com/gfx-rs/wgpu

[39] (2024) llama2.c: Inference llama 2 in one file of pure c. [Online]. Available: https://github.com/karpathy/llama2.c

[40] (2024) llama.cpp: Port of facebook's llama model in c/c++. [Online]. Available: https://github.com/ggerganov/llama.cpp

[41] (2024) Wasmedge-wasinn. [Online]. Available: https://github.com/second-state/WasmEdge-WASINN-examples/tree/master/wasmedge-ggml-llama-interactive#requirement

[42] H. Liu, C. Li, Q. Wu, and Y. J. Lee, "Visual instruction tuning," in *NeurIPS*, 2023.