Document 6 (Source:

https://ec.europa.eu/commission/presscorner/detail/en/QANDA_21_1683)

Why do we need to regulate the use of Artificial Intelligence?

The potential benefits of Artificial Intelligence (AI) for our societies are manifold from improved medical care to better education. Faced with the rapid technological development of AI, the EU decided to act as one to harness these opportunities.

The EU AI Act is the world's first comprehensive AI law. It aims to address risks to health, safety and fundamental rights. The regulation also protects democracy, rule of law and the environment.

While most AI systems will pose low to no risk, certain AI systems create risks that need to be addressed to avoid undesirable outcomes.

For example, the opacity of many algorithms may create uncertainty and hamper the effective enforcement of the existing legislation on safety and fundamental rights. Responding to these challenges, legislative action was needed to ensure a well-functioning internal market for AI systems where both benefits and risks are adequately addressed.

This includes applications such as biometric identification systems or AI decisions touching on important personal interests, such as in the areas of recruitment, education, healthcare, or law enforcement.

Recent advancements in AI gave rise to ever more powerful Generative AI. So-called general-purpose AI models that are being integrated in numerous AI systems are becoming too important for the economy and society not to be regulated. In light of potential systemic risks, the EU puts in place effective rules and oversight.

Which risks will the new AI rules address?

The uptake of AI systems has a strong potential to bring societal benefits, economic growth and enhance EU innovation and global competitiveness. However, in certain cases, the specific characteristics of certain AI systems may create new risks related to user safety and fundamental rights. Some powerful AI models that are being widely used could even pose systemic risks.

This leads to legal uncertainty for companies and potentially slower uptake of AI technologies by businesses and citizens, due to the lack of trust. Disparate regulatory responses by national authorities would risk fragmenting the internal market.

To whom does the AI Act apply?

The legal framework will apply to both public and private actors inside and outside the EU as long as the AI system is placed on the Union market or its use affects people located in the EU.

It can concern both providers (e.g. a developer of a CV-screening tool) and deployers of high-risk AI systems (e.g. a bank buying this screening toolImporters of AI systems will also have to ensure that the foreign provider has already carried out the appropriate conformity assessment procedure, bears a European Conformity (CE) marking and is accompanied by the required documentation and instructions of use.

In addition, certain obligations are foreseen for providers of general-purpose AI models, including large generative AI models.

Providers of free and open-source models are exempted from most of these obligations. This exemption does not cover obligations for providers of general purpose AI models with systemic risks.

Obligations also do not apply to research, development and prototyping activities preceding the release on the market, and the regulation furthermore does not apply to AI systems that are exclusively for military, defence or national security purposes, regardless of the type of entity carrying out those activities.

What are the risk categories?

The Commission proposes a riskbased approach, with four levels of risk for AI systems, as well as an identification of risks specific to general purpose models:

Minimal risk: All other AI systems can be developed and used subject to the existing legislation without additional legal obligations. The vast majority of AI systems currently used or likely to be used in the EU fall into this category. Voluntarily, providers of those systems may choose to apply

the requirements for trustworthy AI and adhere to voluntary codes of conduct.

High-risk: A limited number of AI systems defined in the proposal, potentially creating an adverse impact on people's safety or their fundamental rights (as protected by the EU Charter of Fundamental Rights), are considered to be high-risk. Annexed to the Act is the list of high-risk AI systems, which can be reviewed to align with the evolution of AI use cases.

These also include safety components of products covered by sectorial Union legislation. They will always be considered high-risk when subject to third-party conformity assessment under that sectorial legislation.

Unacceptable risk: A very limited set of particularly harmful uses of AI that contravene EU values because they violate fundamental rights and will therefore be banned:

Social scoring for public and private purposes;

Exploitation of vulnerabilities of persons, use of subliminal techniques;

Real-time remote biometric identification in publicly accessible spaces by law enforcement, subject to narrow exceptions (see below);

Biometric categorisation of natural persons based on biometric data to deduce or infer their race, political opinions, trade union membership, religious or philosophical beliefs or sexual orientation. Filtering of datasets based on biometric data in the area of law enforcement will still be possible;

Individual predictive policing;

Emotion recognition in the workplace and education institutions, unless for medical or safety reasons (i.e. monitoring the tiredness levels of a pilot);

Untargeted scraping of internet or CCTV for facial images to build-up or expand databases.

Specific Transparency risk: For certain AI systems specific transparency requirements are imposed, for example where there is a clear risk of manipulation (e.g. via the use of chatbots). Users should be aware that they are interacting with a machine.

In addition, the AI Act considers systemic risks which could arise from general-purpose AI models, including large generative AI models. These can be used for a variety of tasks and are becoming the basis for many AI systems in the EU. Some of these models could carry systemic risks if they are very capable or widely used. For example, powerful models could cause serious accidents or be misused for far-reaching cyberattacks. Many individuals could be affected if a model propagates harmful biases across many applications.

How do I know whether an AI system is high-risk?

Together with a clear definition of high-risk', the Act sets out a solid methodology that helps

identifying high-risk AI systems within the legal framework. This aims to provide legal certainty for businesses and other operators.

The risk classification is based on the intended purpose of the AI system, in line with the existing EU product safety legislation. It means that the classification of the risk depends on the function performed by the AI system and on the specific purpose and modalities for which the system is used.

Annexed to the Act is a list of use cases which are considered to be high-risk. The Commission will ensure that this list is kept up to date and relevant. Systems on the high-risk list, that perform narrow procedural tasks, improve the result of previous human activities, do not influence human decisions or do purely preparatory tasks are not considered high-risk. However, an AI system shall always be considered high-risk if it performs profiling of natural persons.

What are the obligations for providers of high-risk AI systems?
Before placing a high-risk AI system on the EU market or otherwise putting it into service, providers must subject it to a conformity assessment. This will allow them to demonstrate that their system complies with the mandatory requirements for trustworthy AI (e.g. data quality, documentation and traceability, transparency, human oversight, accuracy, cybersecurity and robustness). This assessment has to be repeated if the system or its purpose are substantially modified.

AI systems being safety components of products covered by sectorial Union legislation will always be deemed high-risk when subject to third-party conformity assessment under that sectorial legislation. Also, for biometric systems a third-party conformity assessment is always required.

Providers of high-risk AI systems will also have to implement quality and risk management systems to ensure their compliance with the new requirements and minimise risks for users and affected persons, even after a product is placed on the market.

High-risk AI systems that are deployed by public authorities or entities acting on their behalf will have to be registered in a public EU database, unless those systems are used for law enforcement and migration. The latter will have to be registered in a non-public part of the database that will be only accessible to relevant supervisory authorities.

Market surveillance authorities will support post-market monitoring through audits and by offering providers the possibility to report on serious incidents or breaches of fundamental rights obligations of which they have become aware. Any market surveillance authority may authorise placing on the market of specific high-risk AI for exceptional reasons.

In case of a breach, the requirements will allow national authorities to have access to the information needed to investigate whether the use of the AI system complied with the law.

What are examples for high-risk use cases as defined in Annex III?

Certain critical infrastructures for instance in the fields of road traffic and the supply of water, gas, heating and electricity;

Education and vocational training, e.g. to evaluate learning outcomes and steer the learning process and monitoring of cheating;

Employment, workers management and access to self-employment, e.g. to place targeted job advertisements, to analyse and filter job applications, and to evaluate candidates;

Access to essential private and public services and benefits (e.g. healthcare), creditworthiness evaluation of natural persons, and risk assessment and pricing in relation to life and health insurance;

Certain systems used in the fields of law enforcement, border control, administration of justice and democratic processes;

Evaluation and classification of emergency calls;

Biometric identification, categorisation and emotion recognition systems (outside the prohibited categories);

Recommender systems of very large online platforms are not included, as they are already covered in other legislation (DMA/DSA).

How are general-purpose AI models being regulated?

General-purpose AI models, including large generative AI models, can be used for a variety of tasks. Individual models may be integrated into a large number of AI systems.

It is important that a provider wishing to build upon a general-purpose AI model has all the necessary information to make sure its system is safe and compliant with the AI Act.

Therefore, the AI Act obliges providers of such models to disclose certain information to downstream system providers. Such transparency enables a better understanding of these models.

Model providers additionally need to have policies in place to ensure that that they respect copyright law when training their models.

In addition, some of these models could pose systemic risks, because they are very capable or widely used.

For now, general purpose AI models that were trained using a total computing power of more than 10^25 FLOPs are considered to carry systemic risks, given that models trained with larger compute tend to be more powerful. The AI Office (established within the Commission) may update this threshold in light of technological advances, and may furthermore in specific cases designate other models as such based on further criteria (e.g. number of users, or the degree of autonomy of the model).

Providers of models with systemic risks are therefore mandated to assess and mitigate risks, report serious incidents, conduct state-of-the-art tests and model evaluations, ensure cybersecurity and provide information on the energy consumption of their models.

For this, they are asked to engage with the European AI Office to draw up Codes of Conduct as the central tool to detail out the rules in cooperation with other experts. A scientific panel will play a central role in overseeing general-purpose AI models.

Why is 10^25 FLOPs an appropriate threshold for GPAI with systemic risks?
This threshold captures the currently most advanced GPAI models, namely OpenAI's GPT-4 and likely Google DeepMind's Gemini.

The capabilities of the models above this threshold are not yet well enough understood. They could pose systemic risks, and therefore it is reasonable to subject their providers to the additional set of obligations.

FLOP is a first proxy for model capabilities, and the exact FLOP threshold can be updated upwards

or downwards by the European AI Office, e.g. in the light of progress in objectively measuring model capabilities and of developments in the computing power needed for a given performance level.

The AI Act can be amended to update the FLOP threshold (by means of a delegated act).

Is the AI Act future-proof?
The Regulation introduces different level of risks and provides clear definitions, including for GPAI.

The legislation sets result-oriented requirements for high-risk AI systems but leaves the concrete technical solutions and operationalisation primarily to industry-driven standards that will ensure that the legal framework is flexible to be adapted to different use cases and to enable new technological solutions.

In addition, the AI Act can be amended by delegated and implementing acts, including to update the FLOP threshold (delegated act), to add criteria for classifying the GPAI models as presenting systemic risks (delegated act), to amend modalities to establish regulatory sandboxes and elements of the real-world testing plan (implementing acts).

How does the AI Act regulate biometric identification?
The use of real-time remote biometric identification in publicly accessible spaces (i.e. facial recognition using CCTV) for law enforcement purposes is prohibited, unless used in one of the following cases:

Law enforcement activities related to 16 specified crimes;
Targeted search for specific victims, abduction, trafficking and sexual exploitation of human beings, and missing persons; or
The prevention of threat to the life or physical safety of persons or response to the present or foreseeable threat of a terror attack.
The list of the 16 crimes contains:

Terrorism;
Trafficking in human beings;
Sexual exploitation of children and child sexual abuse material;

Illicit trafficking in narcotic drugs and psychotropic substances;

Illicit trafficking in weapons, munitions and explosives;

Murder;

Grievous bodily injury;

Illicit trade in human organs and tissue;

Illicit trafficking in nuclear or radioactive materials;

Kidnapping, illegal restraint and hostage-taking;

Crimes within the jurisdiction of the International Criminal Court;

Unlawful seizure of aircraft/ships;

Rape;

Environmental crime;

Organised or armed robbery;

Sabotage, participation in a criminal organisation involved in one or more crimes listed above.

Real-time remote biometric identification by law enforcement authorities would be subject to prior authorisation by a judicial or independent administrative authority whose decision is binding. In case of urgency, authorisation can be done within 24 hours; if the authorisation is rejected all data and output needs to be deleted.

It would need to be preceded by prior fundamental rights impact assessment and should be notified to the relevant market surveillance authority and the data protection authority. In case of urgency, the use of the system may be commenced without the registration.

Usage of AI systems for post remote biometric identification (identification of persons in previously collected video material) of persons under investigation requires prior authorisation by a judicial authority or an independent administrative authority, and notification of the data protection and market surveillance authority.

Why are particular rules needed for remote biometric identification?

Biometric identification can take different forms. It can be used for user authentication i.e. to unlock a smartphone or for verification/authentication at border crossings to check a person's identity against his/her travel documents (one-to-one matching).

Biometric identification could also be used remotely, for identifying people in a crowd, where for

example an image of a person is checked against a database (one-to-many matching).

Accuracy of systems for facial recognition can vary significantly based on a wide range of factors, such as camera quality, light, distance, database, algorithm, and the subject's ethnicity, age or gender. The same applies for gait and voice recognition and other biometric systems. Highly advanced systems are continuously reducing their false acceptance rates.

While a 99% accuracy rate may sound good in general, it is considerably risky when the result leads to the suspicion of an innocent person. Even a 0.1% error rate is a lot if it concerns tens of thousands of people.

How do the rules protect fundamental rights?
There is already a strong protection for fundamental rights and for non-discrimination in place at EU and Member State level, but complexity and opacity of certain AI applications (black boxes') pose a problem.

A human-centric approach to AI means to ensure AI applications comply with fundamental rights legislation. Accountability and transparency requirements for the use of high-risk AI systems, combined with improved enforcement capacities, will ensure that legal compliance is factored in at the development stage.

Where breaches occur, such requirements will allow national authorities to have access to the information needed to investigate whether the use of AI complied with EU law.

Moreover, the AI Act requires that deployers that are bodies governed by public law or private operators providing public services and operators providing high-risk systems to conduct a fundamental rights impact assessment.

What is a fundamental rights impact assessment? Who has to conduct such an assessment, and when?
The use of a high-risk AI system may produce an impact on fundamental rights. Therefore, deployers that are bodies governed by public law or private operators providing public services, and operators providing high-risk systems shall perform an assessment of the impact on fundamental

rights and notify the national authority of the results.

The assessment shall consist of a description of the deployer's processes in which the high-risk AI system will be used, of the period of time and frequency in which the high-risk AI system is intended to be used, of the categories of natural persons and groups likely to be affected by its use in the specific context, of the specific risks of harm likely to impact the affected categories of persons or group of persons, a description of the implementation of human oversight measures and of measures to be taken in case of the materialization of the risks.

If the provider already met this obligation through the data protection impact assessment, the fundamental rights impact assessment shall be conducted in conjunction with that data protection impact assessment.

How does this regulation address racial and gender bias in AI?
It is very important that AI systems do not create or reproduce bias. Rather, when properly designed and used, AI systems can contribute to reduce bias and existing structural discrimination, and thus lead to more equitable and non-discriminatory decisions (e.g. in recruitment).

The new mandatory requirements for all high-risk AI systems will serve this purpose. AI systems must be technically robust to guarantee that the technology is fit for purpose and false positive/negative results are not disproportionately affecting protected groups (e.g. racial or ethnic origin, sex, age etc.).

High-risk systems will also need to be trained and tested with sufficiently representative datasets to minimise the risk of unfair biases embedded in the model and ensure that these can be addressed through appropriate bias detection, correction and other mitigating measures.

They must also be traceable and auditable, ensuring that appropriate documentation is kept, including of the data used to train the algorithm that would be key in ex post investigations.

Compliance system before and after they are placed on the market will have to ensure these systems are regularly monitored and potential risks are promptly addressed.

When will the AI Act be fully applicable?

Following its adoption by the European Parliament and the Council, the AI Act shall enter into force on the twentieth day following that of its publication in the official Journal. It will be fully applicable 24 months after entry into force, with a graduated approach as follows:

6 months after entry into force, Member States shall phase out prohibited systems;

12 months: obligations for general purpose AI governance become applicable;

24 months: all rules of the AI Act become applicable including obligations for high-risk systems defined in Annex III (list of high-risk use cases);

36 months: obligations for high-risk systems defined in Annex II (list of Union harmonisation legislation) apply.

How will the AI Act be enforced?

Member States hold a key role in the application and enforcement of this Regulation. In this respect, each Member State should designate one or more national competent authorities to supervise the application and implementation, as well as carry out market surveillance activities.

To increase efficiency and to set an official point of contact with the public and other counterparts, each Member State should designate one national supervisory authority, which will also represent the country in the European Artificial Intelligence Board.

Additional technical expertise will be provided by an advisory forum, representing a balanced selection of stakeholders, including industry, start-ups, SMEs, civil society and academia.

In addition, the Commission will establish a new European AI Office, within the Commission, which will supervise general-purpose AI models, cooperate with the European Artificial Intelligence Board and be supported by a scientific panel of independent experts.

Why is a European Artificial Intelligence Board needed and what will it do?

The European Artificial Intelligence Board comprises high-level representatives of competent national supervisory authorities, the European Data Protection Supervisor, and the Commission. Its role is to facilitate a smooth, effective and harmonised implementation of the new AI Regulation.

The Board will issue recommendations and opinions to the Commission regarding high-risk AI

systems and on other aspects relevant for the effective and uniform implementation of the new rules. Finally, it will also support standardisation activities in the area.

What are the tasks of the European AI Office?

The AI Office has as its mission to develop Union expertise and capabilities in the field of artificial intelligence and to contribute to the implementation of Union legislation of artificial intelligence in a centralised structure.

In particular, the AI Office shall enforce and supervise the new rules for general purpose AI models. This includes drawing up codes of practice to detail out rules, its role in classifying models with systemic risks and monitoring the effective implementation and compliance with the Regulation. The latter is facilitated by the powers to request documentation, conduct model evaluations, investigate upon alerts and request providers to take corrective action.

The AI Office shall ensure coordination regarding artificial intelligence policy and collaboration between involved Union institutions, bodies and agencies as well as with experts and stakeholders. In particular, it will provide a strong link with the scientific community to support the enforcement, serve as international reference point for independent experts and expert organisations and facilitate exchange and collaboration with similar institutions across the globe.

What is the difference between the AI Board, AI Office, Advisory Forum and Scientific Panel of independent experts?

The AI Board has extended tasks in advising and assisting the Commission and the Member States.

The AI Office is to be established within the Commission and shall work to develop Union expertise and capabilities in the field of artificial intelligence and to contribute to the implementation of Union legislation of artificial intelligence. Particularly, the AI Office shall enforce and supervise the new rules for general purpose AI models.

The Advisory Forum will consist of a balanced selection of stakeholders, including industry, start-ups, SMEs, civil society and academia. It shall be established to advise and provide technical expertise to the Board and the Commission, with members appointed by the Board among stakeholders.

The Scientific Panel of independent experts supports the implementation and enforcement of the Regulation as regards GPAI models and systems, and the Member States would have access to the pool of experts.

What are the penalties for infringement?

When AI systems are put on the market or in use that do not respect the requirements of the Regulation, Member States will have to lay down effective, proportionate and dissuasive penalties, including administrative fines, in relation to infringements and communicate them to the Commission.

The Regulation sets out thresholds that need to be taken into account:

Up to 35m or 7% of the total worldwide annual turnover of the preceding financial year (whichever is higher) for infringements on prohibited practices or non-compliance related to requirements on data;

Up to 15m or 3% of the total worldwide annual turnover of the preceding financial year for non-compliance with any of the other requirements or obligations of the Regulation, including infringement of the rules on general-purpose AI models;

Up to 7.5m or 1.5% of the total worldwide annual turnover of the preceding financial year for the supply of incorrect, incomplete or misleading information to notified bodies and national competent authorities in reply to a request;

For each category of infringement, the threshold would be the lower of the two amounts for SMEs and the higher for other companies.

In order to harmonise national rules and practices in setting administrative fines, the Commission, counting on the advice of the Board, will draw up guidelines.

As EU Institutions, agencies or bodies should lead by example, they will also be subject to the rules and to possible penalties; the European Data Protection Supervisor will have the power to impose fines to them.

What can individuals do that are affected by a rule violation?

The AI Act foresees a right to lodge a complaint with a national authority. On this basis national authorities can launch market surveillance activities, following the procedures of the market

surveillance regulations.

Additionally, the proposed AI Liability Directive aims to provide persons seeking compensation for damage caused by high-risk AI systems with effective means to identify potentially liable persons and obtain relevant evidence for a damage claim. For this purpose, the proposed Directive provides for the disclosure of evidence about specific high-risk AI systems that are suspected of having caused damage.

Moreover, the revised Product Liability Directive will ensure that compensation is available to individuals who suffer death, personal injury or property damage that is caused by a defective product in the Union and clarify that AI systems and products that integrate AI systems are also covered by existing rules.

How do the voluntary codes of conduct for high-risk AI systems work?
Providers of non-high-risk applications can ensure that their AI system is trustworthy by developing their own voluntary codes of conduct or adhering to codes of conduct adopted by other representative associations.

These will apply simultaneously with the transparency obligations for certain AI systems.

The Commission will encourage industry associations and other representative organisations to adopt voluntary codes of conduct.

How do the codes of practice for general purpose AI models work?
The Commission invites providers of general-purpose AI models and other experts to jointly work on a code of practice.

Once developed and approved for this purpose, these codes can be used by the providers of general-purpose AI models to demonstrate compliance with the relevant obligations from the AI Act, following the example of the GDPR.

This is especially relevant to detail out the rules for providers of general-purpose AI model with systemic risks, to ensure future-proof and effective rules for risk assessment and mitigation as well

as other obligations.

Does the AI Act contain provisions regarding environmental protection and sustainability?
The objective of the AI proposal is to address risks to safety and fundamental rights, including the fundamental right to a high-level environmental protection. Environment is also one of the explicitly mentioned and protected legal interests.

The Commission is asked to request European standardisation organisations a standardisation deliverable on reporting and documentation processes to improve AI systems resource performance, such as reduction of energy and other resources consumption of the high-risk AI system during its lifecycle, and on energy efficient development of general-purpose AI models.

Furthermore, the Commission by two years after the date of application of the Regulation and every four years thereafter, is asked to submit a report on the review of the progress on the development of standardisation deliverables on energy efficient development of general-purpose models and asses the need for further measures or actions, including binding measures or actions.

In addition, providers of general purpose AI models, which are trained on large data amounts and therefore prone to high energy consumption, are required to disclose energy consumption.

The Commission is asked to develop an appropriate methodology for this assessment.

In case of general purpose AI models with systemic risks, energy efficiency furthermore needs to be assessed.

How can the new rules support innovation?
The regulatory framework can enhance the uptake of AI in two ways. On the one hand, increasing users' trust will increase the demand for AI used by companies and public authorities. On the other hand, by increasing legal certainty and harmonising rules, AI providers will access bigger markets, with products that users and consumers appreciate and purchase. Rules will apply only where strictly needed and in a way that minimises the burden for economic operators, with a light governance structure.

The AI Act further enables the creation of regulatory sandboxes and real world testing, which provide a controlled environment to test innovative technologies for a limited time, thereby fostering innovation by companies, SMEs and start-ups in compliance with the AI Act. These, together with other measures such as the additional Networks of AI Excellence Centres and the Public-Private Partnership on Artificial Intelligence, Data and Robotics, and access to Digital Innovation Hubs and Testing and Experimentation Facilities will help build the right framework conditions for companies to develop and deploy AI.

Real world testing of High-Risk AI systems can be conducted for a maximum of 6 months (which can be prolonged by another 6 months). Prior to testing, a plan needs to be drawn up and submitted it to the market surveillance authority, which has to approve of the plan and specific testing conditions, with default tacit approval if no answer has been given within 30 days. Testing may be subject to unannounced inspections by the authority.

Real world testing can only be conducted given specific safeguards, e.g. users of the systems under real world testing have to provide informed consent, the testing must not have any negative effect on them, outcomes need to be reversible or disregardable, and their data needs to be deleted after conclusion of the testing. Special protection is to be granted to vulnerable groups, i.e. due to their age, physical or mental disability.

Besides the AI Act, how will the EU facilitate and support innovation in AI?

The EU's approach to Artificial Intelligence is based on excellence and trust, aiming to boost research and industrial capacity while ensuring safety and the protection of fundamental rights. People and businesses should be able to enjoy the benefits of AI while feeling safe and protected. The European AI Strategy aims at making the EU a world-class hub for AI and ensuring that AI is human-centric and trustworthy. In April 2021, the Commission presented its AI package, including: (1) a review of the Coordinated Plan on Artificial Intelligence and (2) its proposal for a regulation laying down harmonised rules on AI.

With the Coordinated Plan on AI the European Commission has adopted a comprehensive strategy to promote the development and adoption of AI in Europe. It focuses on creating enabling conditions for AI development and uptake, ensuring excellence thrives from the lab to the market, increasing the trustworthiness of AI, and building strategic leadership in high-impact sectors.

The Commission aims to leverage the activities of Member States by coordinating and harmonizing their efforts, to foster a cohesive and synergistic approach towards AI development and adoption. The Commission also put in place the European AI Alliance platform, which brings together stakeholders from academia, industry, and civil society to exchange knowledge and insights on AI policies.

Moreover, the Coordinated plans foresees several measures that aim to unlock data resources, foster critical computing capacity, increase research capacities, support a European network of Testing and Experimentation Facilities (TEFS) and support SMEs through European Digital Innovation Hubs (EDIHs).

What is the international dimension of the EU's approach?
The AI Act and the Coordinated Plan on AI are part of the efforts of the European Union to be a global leader in the promotion of trustworthy AI at international level. AI has become an area of strategic importance at the crossroads of geopolitics, commercial stakes and security concerns.

Countries around the world are choosing to use AI as a way to signal their desires for technical advancement due to its utility and potential. AI regulation is only emerging and the EU will take actions to foster the setting of global AI standards in close collaboration with international partners in line with the rules-based multilateral system and the values it upholds. The EU intends to deepen partnerships, coalitions and alliances with EU partners (e.g. Japan, the US, India, Canada, South Korea, Singapore, or the Latin American and Caribbean region) as well as multilateral (e.g. OECD, G7 and G20) and regional organisations (e.g. Council of Europe).

*Updated on 14/12/2023

Document 7 (Source: https://bg3.wiki/wiki/The_Emperor)

The Emperor is a mind flayer who appears in Baldur's Gate 3. It[note 1] plays a key role in the main story, but its identity is intentionally obscured until later parts of the game, allowing the player to ultimately decide for themselves if they want to know more about it, and whether or not it is trustworthy.

Overview

Identity

The Emperor plays a key role in the main story of Baldur's Gate 3, and as part of this role its identity and personal background are kept obfuscated for much of the game. It very carefully divulges information that it deems necessary, sometimes arguing that the player is not ready for the answer yet, or that it will reveal specific information in the future.

During Acts One and Two, the Emperor only "meets" with the player as the Dream Guardian. At the beginning of Act Three, the player finally meets the Emperor face to face, an event which reveals

that it is a mind flayer.

Through all three Acts, the Emperor generally serves as a guide, and unlikely ally to the party, having the means to protect their minds from the influence of the Absolute, through the use of the prisoner within the Astral Prism.

"Don't let my form deceive you. I am the one that's been protecting you. I am the one that came to you in your dreams. Help me.
The Emperor, during Act 3

Personal quest

After reaching the Elfsong Tavern in Act Three, the Emperor will initiate the quest Visit the Emperor's Old Hideout, in which the player can better get to know the Emperor. It discloses some of its past, during its time in the city and from before it became illithid.

Recruitment

The Emperor can appear in multiple combat encounters as a controllable ally, a neutral ally, or an enemy. It cannot, however, become a full member of the player's party or camp.

Romance

The Emperor can have a romance with the player during Act Three. See Romance.

History

Details about the Emperor's personal history are intentionally obfuscated during most of the game, but the player has the opportunity to learn more about it through conversations, interactions with other characters, reading books, and completing specific side quests.

Ico knownSpells lvl 03.png Act 3 Spoilers! This section reveals details about the story of Baldur's Gate 3.

An Adventurer, I came from Baldur's Gate, though I was never one to be constrained by circumstance. I longed for more.

That longing brought me to Moonrise Towers on a search for treasure. To a colony of mind flayers who caught me and changed me.
The Emperor was once Balduran, an adventurer who founded a coastal village called Grey Harbour. After securing enough money to fund the building of the Wall that led to Baldur's Gate being founded, he felt the call of the sea once more. On the voyage, and following a shipwreck, Balduran made his way to Moonrise Towers in search of fortune. There, he found a coven of mind flayers who infected him with an illithid tadpole. As a record of his interrogation by Enver Gortash during the planning phases of the Absolute Hoax states, he spent ten years under the thrall of the Moonrise Elder Brain.

After Balduran was reborn as an illithid and broke free from the Elder Brain the Absolute, it returned to Baldur's Gate, living in the shadows and feeding on the brains of criminals. Initially struggling with its identity as a mind flayer, Balduran eventually embraced its new form.

Balduran's new acceptance of its illithid form caused a wedge to form between it and its close companion, the dragon Ansur. Ansur attempted to kill Balduran as it slept, believing this would be a merciful death. The Emperor sensed the attempt, and in its struggle to protect itself from being murdered, it killed Ansur in self-defence. [1]

After Ansur's death, Balduran came to be called the Emperor as it used its newfound psychic influence to rule Baldur's Gate from the shadows. For the next four centuries, it made its haven under the Elfsong tavern, keeping various sentimental knick knacks from its time as Balduran.

I had the fortune of meeting Duke Stelmane. We formed a partnership
During those four centuries, it also came to be associated with the Knights of the Shield, a lawful and neutral evil conglomerate of politicians and merchants manipulating events behind the scenes. Duke Stelmane was a major figure of this secret society, acting as the Emperor's envoy while it secretly kept her enthralled. [note 2]

Sometime before the events of the game, Enver Gortash and the Dark Urge captured the Emperor, and brought it back under the thrall of the Moonrise Elder Brain, who was now wearing the Crown of Karsus and had become the Netherbrain masquerading as the Absolute. The Netherbrain, sought to

have all three Chosen of the Dead Three killed, and specifically picked the Emperor, unbeknown to it, to lead a team of illithids on a nautiloid to search for and steal from the Githyanki the Astral Prism containing their prince, Orpheus.[2]

Events of Baldur's Gate 3

Act Two finale

Main article: Help Your Protector

On the way to Baldur's Gate, the party will be ambushed by a group of Gish'ra warriors while resting at Wyrm's Lookout. Entering the portal to the Astral Prism, the party will hear their Dream Guardian calling out for help. However, when the party reaches them, it is only to discover that the true identity of their visitor is the illithid known as the Emperor.

After defending the Emperor, it will explain how it used the power of the Prism and Orpheus to protect the party from the Absolute, and recite to the party its history as an adventurer and finding freedom from the Absolute. The Emperor will offer the party an Astral Touched Tadpole, which causes the user to transform into a partial-illithid. It insists the path of the mind flayer is preferable, regardless of the player's view on them.

Though this may seen contradictory to its previous promise as the Dream Guardian; to ensure the party do not become mind flayers, this promise refers to the player becoming a mind flayer unwillingly because of the Elder Brain. The Emperor is in favour of the player becoming a mind flayer of their own volition and without the influence of the Elder Brain.

Act Three

Elfsong Tavern

Main article: Visit the Emperor's Old Hideout

As the party nears the Elfsong, the Emperor will remark that the tavern is the location of its old hideout. The hideout proper is in the basement, past the Knights of the Shield's hideout. In it, the player will find various sentimental knick knacks from the Emperor's previous life, before becoming an illithid.

Around the room is its old dog Rascal's collar, its favourite recipe (fiddlehead soup), its first adventuring sword, and part of a cutlery set from its mother; the butter knife having been lost during its last shipwreck on the Isle of Balduran, inside the wreck of the Wandering Eye ship.

There are also some more illithid-adequate items such as chains for its preferred prey - allegedly criminals and lawbreakers - and jars for brains.

The Wyrmway

See also: Wyrmway and The Blade of Frontiers

Once the party completes the Wyrmway trials, they will find the corpse of Ansur the Dragon. Interacting with his body will awaken Ansur's spirit, which briefly possesses the player in order to communicate. As Ansur's introduction concludes, he will detect the Emperor within the Astral Prism.

Ico knownSpells lvl 03.png Act 3 Spoilers! This section reveals details about the story of Baldur's Gate 3.

Ansur will reveal that the Emperor in fact was formerly Balduran, the founder of Baldur's Gate. Furthermore, he explains that while the Emperor initially did not want to become a mind flayer, it eventually fully embraced its new form, and its comfort with this caused a rift between the Emperor and Ansur. After "exhausting all possibility of reversing (the Emperor's) condition", Ansur was agonizing and the Emperor (as seen in the letter on Ansur's body) attempted to convince him to leave. Ansur then attempted to murder the Emperor during its sleep as a mercy killing, and the Emperor killed Ansur in self-defense.

This development is somewhat foreshadowed when the player first meets The Emperor in their true form, as the song that plays during the encounter is a variation of The Elf Song, which prominently features Balduran in its lyrics.

Endings

Ico knownSpells lvl 03.png Act 3 Spoilers! This section reveals details about the story of Baldur's Gate 3.

Let the Emperor use the Netherstones

The Emperor unless convinced otherwise is mostly concerned with its survival and prosperity. Should the player allow it to wield the Netherstones, it will follow through on destroying the Elder Brain, at the cost of letting it "assimilate" with Orpheus.

If the player suggests to the Emperor to take control of the Netherbrain, it will mention that the

thought of becoming the Absolute did cross its mind. But unless otherwise persuaded, it will refuse, claiming that whoever becomes the leader of the Cult of the Absolute will be in an open war with the Githyanki, which is a war it is not certain it will survive. The Emperor will destroy the Netherbrain, and the parasites within its control in this ending.

The Emperor controls the Netherbrain

It is also possible, after suggesting it to take control of the Netherbrain, to persuade it. In this scenario, it does not free the player or their party, instead making them mindless thralls and assuming absolute control of them, continuing the Grand Design.

Orpheus is freed

If the player frees Orpheus, the Emperor will abandon the party, and side with the Netherbrain for the sake of its own survival, as it believes that Orpheus will kill it.

Attack the Emperor

The Emperor can be attacked and killed when it first reveals itself to be a mind flayer. This will result in the influence of the Netherbrain taking over control of the party, ending the game.

List of interactions

See Dream Guardian to read about its previous conversations with the player when it was in disguise.

Charm Person Icon.png Romance Spoilers This section reveals details about romance and may contain mature themes.

Players have a limited number of opportunities to interact with the Emperor, and as such, opportunities for conversation are much more limited compared to that of companions.

Conversation scenes are available, but only occur during Act 3, after its "true" identity is revealed to the player, and all scenes require a long rest to trigger. The Emperor will occasionally also talk to the player as they walk through different locations in Baldur's Gate.

Conversation scenes

Known conversation opportunities with the Emperor currently include the following cases, but each scene appears to have multiple outcomes that affect the tone of all subsequent conversations.

Depending on the player's choices, the Emperor's behaviour has many possible states. The more the player treats the Emperor like a "person", the more it will act as such, compared to other illithids.

The more the player treats The Emperor like a monstrosity with hostile intent, the more it will respond to the player with threatening language and visions of it acting like a hostile illithid.

Identity revealed

During Help Your Protector at the start of Act 3, a conversation is automatically triggered when the player ventures far enough into the Astral Plane. A combat encounter in some form is inevitable from this conversation, and then another set of conversation options are available after the combat resolves. The Emperor will have nothing further to say when this conversation ends, even if the player tries to interact with it further.

Regarding Duke Stelmane

When the player first explores the Rivington area, being in proximity to certain characters or objects will "inform" the player about the recent death of Duke Belynne Stelmane. This will trigger a line of ambient commentary from The Emperor. The next time a Long Rest is triggered, the player may trigger a scene discussing The Emperor's reactions in more depth. Certain dialogue choices made during earlier conversations seem to disqualify the player from this scene. If the player does not long rest before completing the quest Visit the Emperor's Old Hideout, this scene will be skipped entirely.

On conclusion of Visit the Emperor's Old Hideout

This scene may be available to trigger (by long resting) after the player completes the quest Visit the Emperor's Old Hideout.

Possible states for this scene appear to vary heavily depending on the player's choices in prior conversation scenes, with the general differentiating factor being the "attitude" the player appears to express towards illithids, and towards the Emperor, through their selected options in these prior scenes.

If the player tried to kill the Emperor in Act One, by choosing the dialogue option "You do a great impression of a human. But you're not fooling me." , the Emperor offers to share memories through a vision. This vision shows Stelmane paralysed in pain, being brainwashed, and turning into the Emperor's puppet. Her face emotionless, and the Emperor puppeteering her gestures to get a sense of company. Such was its true relationship with Duke Stelmane. [note 2]

The Emperor uses this memory to frighten the player. It gives them orders, and threatens to make them half-illithid even if they refuse.

Romance

In terms of game mechanics, it is technically possible to romance the Emperor. [note 3]

If the player chooses to reject its advances, the Emperor's attitude in conversation will change in a way that appears to be reactively appropriate to the way it was treated. For example, if the option "Absolutely not, you freak!" is chosen at any opportunity, the Emperor's treatment of the player takes a much more hostile tone in all future interactions.

Players have a limited number of opportunities to interact with the Emperor, and as such, opportunities for romantically-styled interactions are much more limited compared to the other primary companions.

If the player visits Crèche Y'llek prior to the start of Act 3, killing the Dream Guardian will subsequently lock the player out of romancing the Emperor, and from interacting with it in general.

There are many possible ways to interact with the Emperor in the available conversation scenes. It currently seems that the primary way to unlock "romantic" options is by choosing dialogue that generally treats the Emperor more like "any other person", and does not show explicit hostility towards its actions, or its illithid characteristics.

The player does not need to accept the powers of the Astral-Touched Tadpole to unlock this option. The Emperor seems to take offence to destroying the tadpole, but more testing is needed to determine if this has any effect on the available scenes.

The scene that occurs after completing Visit the Emperor's Old Hideout is generally regarded as the "primary" romantic scene. As long as the player is receptive to the Emperor's advances, conclusions to this scene will allow the player to engage in more intimate activities with it.

Conversation options that acknowledge this romance (after the primary scene has concluded) appear to exist in a limited number of places. For example, it is possible to tell Raphael "I don't want

any part of this the Emperor is my lover." during a specific conversation, if initiated after the romance scene has happened.

Engaging in the primary scene has no effect on other ongoing romances, even when romancing Lae'zel, who is generally hostile to illithids.

Achievements

A-Mind Blown.jpg

Mind Blown

Romance the Emperor.

Gallery

They called me The Emperor

They called me The Emperor

The Emperor feeding on criminals

The Emperor feeding on criminals

Character portrait by Edward Vanderghote

Character portrait by Edward Vanderghote

The Emperor's model

The Emperor's model

Notes

The Emperor's existence confirms the Dream Guardian as being an illithid influence, albeit in a different way.

In Early Access, the Dream Guardian (known then as Dream Visitor) was implied to be a mental manifestation of the player's tadpole, as it eased them towards using their powers more, as well as

showing them a future of domination and control.

In the Full Release, the Emperor plays a similar role, in the sense that it also encourages the player to expand their potential through using the tadpole's power, but it is much more passive. In addition, its interests seem to be aligned against the Absolute.

Footnotes

The Emperor, like other mind flayers, is addressed using the "it" pronoun. It is incidentally referred to as "he" in-game, and "they" in the game's files, possibly due to an oversight, or characters conflating its current and previous identities.

The Emperor's vision of its control over Belynne Stelmane is corroborated by the 5e module, Baldur's Gate: Descent into Avernus. In it, Stelmane is described as having a secret, mental battle against a mind flayer. This mind flayer is very likely the Emperor itself, and as a result, puts its entire "alliance" with Stelmane into question. It is very possible the Emperor and Stelmane did not have a proper alliance at all, and rather, the Emperor enthralled her for its needs. Whether this was always the case, or if they had a genuine alliance beforehand, isn't fully clear.

This romance behaves somewhat differently from that with companions, as the Emperor generally cannot be interacted with outside of cutscenes, and romantic progression is limited to the final act of the game.

References

Dialogue with Ansur.

The Netherbrain's dialogue to the player at the Morphic Pool.

Document 8 (Source: https://whattocook.substack.com/p/so-into-northern-spain)

so into northern spain!

our magical urban-plus-outdoor-adventure itinerary

CAROLINE CHAMBERS

MAY 29, 2024

I recently returned from a trip to Northern Spain with two of my best friends, Lily and Nellie, sponsored by elsewhere, a travel company that works with local experts to create truly unique travel experiences.

On our last day in Spain, we had an incredible private tour of the Guggenheim Museum Bilbao, and then partook in our favorite trip ritual: hopping around from pintxo bar to pintxo bar (pintxos = small little plates of food similar to tapas but they are typically sitting on the bar and you point to what you

want), drinking txakoli (a lightly effervescent white wine that is very, very popular in the region), and playing gin rummy.

We then headed back to our hotel room, crammed onto the bed with, well, another bottle of txakoli, and recorded what I can only assume was the greatest podcast episode of all time. We recapped the full trip, what we loved, what we would change. We shared the most absurd moments. We laughed so hard and made so much fun of each other in a way that only really old friends who have just spent 12 days straight together can do. Ive been so excited to share it.

But the recording failed. I have no idea what happened, but I blame the txakoli.

I thought about re-recording with them remotely, but the magic of that moment, being there together on the last night of a truly perfect trip, is gone. So instead, Ill share the highlights here! Full itinerary with ALL the details is here.

We spent two nights in Madrid and could have spent at least four. Theres so much to see and do here, and we barely scratched the surface. Our highlights were shopping in the Chueca and Salamanca neighborhoods Malababa, Soeur, and Sessùn were some of our favorite shops. We had a wonderful meal at Charrúa Madrid and fun cocktails at Ficus Bar. You can find all of our favorite shops, tapas bars, and many places that we didnt get to explore but wanted to on my Spain map here.

strolling the charming, windy streets of chueca!

San Sebastián is the coolest town worth a trip to Spain all on its own. Its a coastal town on the Northern coast of Spain, right next to the French border, and it fully charmed us. Its that perfect European blend of old meets new hip young surfers going to 200-year-old pintxo bars to meet up with their friends after surfing all afternoon, 100-year-old pastry shops next door to chic Spanish design ateliers. We spent three nights there and loved every minute, but these were our highlights:

cooking in a private gastronomic club

When youre wandering the cobblestone streets of San Sebastián looking for your next pintxo, you might find yourself stumbling through the doorway of a friendly looking restaurant, only to be turned away. Disculpa, privado! theyll tell you Sorry, private!

Youve stumbled into a sociedad gastronómica one of San Sebastiáns 100+ private gastronomic clubs. Each one has its own personality and offering, but at its core is this: its a private social club centered around cooking and eating with fellow members. Members reserve a time to cook depending on the size of the club, three to four members can cook at a time and they can invite guests to join them. Members will head to one of the many local markets, grab their food, and then bring it back to the shared kitchen to cook for/with their friends and family.

post-lunch txakoli on the deck at the club

The only hitch? You have to be with a member to go inside of one! This was the coolest tour that elsewhere set up for us Jani, our guide, was an incredibly cool young mom and entrepreneur (she owns her own tour company) and is a member of several clubs. She picked us up at our hotel, took us to her favorite local market and farmers market to grab ingredients, and then back to her club, which was a stone building with wooden beams and felt like a scene out of a Game of Thrones banquet. We cooked a local fish dish, a tomato salad, and seared white asparagus, and she taught us all about Basque cuisine and the local gastronomic clubs.

We spent the entire afternoon cooking and eating and drinking txakoli on the deck of her club overlooking the city. A cannot miss experience if you find yourself in San Seb! Heres a great Saveur article if you want to read more about the clubs.

pintxo hopping

Pick three or four pintxos bars that are close together, and hop from bar to bar, eating a snack and having a glass of txakoli at each one. This is always my favorite way to explore a new place on limited time an appetizer at one spot, a meal at another, after-dinner drinks at a third, so the pintxos culture really allows you to see a lot of places in one night!

My map has a lot of great pintxos bars saved, but heres an especially great hop (these are all on my map):

Start at Ganbara and grab whatever looks good we had delightful fried padrón peppers.

Walk over to Txepetxa Taberna for a little sandwich my favorites were the simple ones filled with local jámon and idiazábal cheese (similar to manchego).

The cheesy risotto at Borda Berri was phenomenal.

Finish your night at Otaegui, the oldest bakery in the city, for pantxineta, a delightful puff pastry and custard-filled local treat, and a slice of Basque cheesecake.

pintxo hopping around the old town!

surfing

Nellie is a big surfer and after a full 24 hours of pestering, finally convinced me to go out with her. There are tons of surf shops that rent wetsuits and boards lining the beach, so it was really easy to get into the water. The waves were absolutely perfect. Gentle, clear water, no getting-stuck-in-a-washing-machine vibes when you fall of your board. It was so, so much fun.

My favorite part of the trip was our four days on the camino. The Camino de Santiago, also known as the Way of Saint James, is a well-trod pilgrimage route that traverses Spain, culminating at the Cathedral of Santiago de Compostela in Galicia. The camino has origins back to the medieval era and has four routes connecting different parts of Spain to Galicia. The routes consist of trails, modern roads, and original medieval stone pathways

that pass through everything from modern city to medieval village

to small fishing town

to stunning wilderness landscapes.

The flysch was truly breathtaking. If you wind up in San Sabastián and want to walk just one day of the camino, walk from Orio to Zumaia, all the way out past the town to see the flysch, then sleep at Hotel Iturregi that night!
The Camino Francés is the most popular route, but my friends and I did a small portion of the Camino Norte, which San Sebastián is right on. The Norte is challenging and stunningly beautiful, with steep, rugged terrain, but with plenty of stopping places for pintxos and a glass of wine along the way. Its hard to sum up what a wonderful experience hiking it was.

We originally wanted to hike directly from one location to the next to hike from one hotel to the next without ever getting in a car 15 to 20 miles per day on the camino. However, the hotels along the route are rustic, more rustic than we were up for, so we wound up switching to stay at a gorgeous hotel in a txakoli vineyard, and just took cabs to and from the camino for our last two days.

Hiking the camino was a bucket list experience we hiked through fishing villages and past a truly breathtaking flysch rock formation, a cult tried to recruit us by beckoning us in with warm tea and cake, we talked about everything, we talked about nothing, we laughed so hard it hurt. I cannot wait to get back to the camino one day.

Bilbao was an excellent urban re-introduction at the end of four days on the camino. We really didnt know what to expect, and we were pleasantly surprised. I wouldnt make a trip to Spain specifically for Bilbao, but Id certainly make an effort to visit if I was in Northern Spain.

The Guggenheim alone could occupy an entire day its a massive collection of modern and contemporary art. The architecture itself is reason alone to visit. We loved touring the museum with a private tour guide who regaled us with detailed information about every single piece.

We strolled along the river from the Guggenheim all the way to the seven streets area, where we

bopped between shops and pintxo bars for the rest of the day. Admittedly, we enjoyed a lot of Italian food in Bilbao as we were feeling a little burnt out on Basque food by this point in the trip. No shame in dabbling in a little pizza while in Spain! Youll find lots of Basque and Italian recs on my map!

After two nights in Bilbao, we headed home. It was a truly magical trip that well still be laughing and reminiscing about when were old ladies.

I love electrolyte powders but so many brands have a lot of sodium in them, which can make me feel really bloated. Ultima is lower-sodium and the flavors are great!

Lily had a lanyard strap iPhone case that I was really jealous of. Kind of dorky looking but who cares, it was so convenient to have her phone handy for photos and looking up directions!

I will never stop yappin about my Owala water bottle. Having the ability to drink out of the straw or tilt it and chug it never gets old!

I brought disposable cameras for each of us and I cant wait to get the film back!! Mattis saw them and has been begging me to get him one it would be fun to give your kids a disposable camera to capture a summer trip!

Pan-fried dover sole with buttery tomato sauce and corn risotto. The reviews on this weeks recipe are RAVING! It really is so good and fancy feeling. You have to make this one ASAP, especially if you travel somewhere with access to great fish this summer!

Sunshine curry was a cult fave last summer of yours and of mine! I love making a huge batch and eating it for lunch all week long (a perk of the fact that my kids wont touch it, though I know many kids love this one!).

My summer tomato galette is the stuff of summer produce dreams. Buy a store-bought pie crust to

make it even easier, but its worth making the crust from scratch if you have 15 extra minutes! PS: Make the boursin summer squash galette while youre at it. Two summery galettes + a simple salad of romaine tossed with salt, pepper, really good extra-virgin olive oil, and balsamic, with shaved Parm, and chopped almonds would be such a chic little meal.

Chicken panzanella is one of my all-time faves. Crisp croutons, juicy tomatoes, perfectly cooked chicken, balsamic, olive oil, basil, with big hunks of melty goat cheese. Its the perfect summer lunch or dinner! Dont let your croutons burn!

I loved reading about all of your happy places in the comments of last weeks post! My number one takeaway is that I need to spend more time in Maine so many of your happy places are there! Including this weeks winner, Heather.

Dreamy! And now you can bring your new pair of Lake Pajamas to Boothbay Harbor this summer! Email me your size and address!

Do you have any Spain recs to add? Anyone taking an exciting trip this summer? Were staying local for most of the summer so that we can enjoy our new cabin, minus our annual two-week Southern voyage to DeBordieu, SC to visit Georges family and Bald Head Island, NC to visit mine!

Document 9 (Source: https://dmtalkies.com/the-zone-of-interest-ending-explained-and-summary-2023-film/)

The Zone Of Interest Ending Explained & Film Summary: What Happens To Rudolf And Hedwig Hoss?

PUBLISHED

FEBRUARY 21, 2024

BY

SOURYA SUR ROY

0COMMENTS

The Zone Of Interest Ending Explained Film Summary Hedwig Ross, Rudolph Ross

Credits: A24

The Zone of Interest is a new historical drama film by English filmmaker Jonathan Glazer that manages to recreate a terrible moment from history with a unique and devastating effect. Loosely adapted from Martin Amis novel of the same name, the films plot follows the Hoss family, who live right beside the Auschwitz concentration camp, going about their usual lives with no concern for the terrible crimes being committed right outside. The Zone of Interest is all about subtle, indirect expressions that are poignant enough to pierce through the visual layer, successfully making the viewer all the more uncomfortable with every passing minute.

Spoiler Alert

Plot Summary: What Is The Film About?

The Zone of Interest opens with a noticeably long black screen, with only a soft sound being eerily stretched in the background, perhaps preparing us for what is to unfold on screen over the next hundred or so minutes. When the visuals come on, though, there is nothing unusual or out of the ordinary, as a family is seen spending some personal time by the forested banks of a river. This is a secluded spot reserved only for the family, and it seems to be their most common way of spending leisure time. As the girls are led by a nanny through the bushes, possibly for some lesson in gardening and wildlife, the boys jump into the river along with their father. Sometime later, the family reunites and leaves the riverbank, driving away in two black, sinister-looking cars. On that very night, the father of the house is seen going around, switching off all the lights, before going to bed.

While there is really nothing odd in this whole presentation of a family spending a day with themselves, the chilling reality of the matter is revealed when the film introduces the particular lot. The family is that of Rudolf Hoss, a notorious real figure from history, infamous for being a distinguished SS officer and the commander of the Auschwitz concentration camp. Most of the entire film, and the whole of the opening scene, actually takes place in Auschwitz, meaning that the leisurely picnic of the big family literally took place only a few miles away from the spot of the ongoing genocide. This is the very premise of The Zone of Interest, for it shows the tumultuous time of history from the perspective of the Hoss family, mainly the patriarch Rudolf and his wife, Hedwig.

The couple lives in an idyllic resort with their two sons and three daughters, the youngest still a baby, right on the other side of the high walls of the concentration camp. Despite the inhuman

torture and killing going on right outside the walls that separate their lives, the Hoss family members are not perturbed by the matter at all. Instead, they are rather accustomed to Auschwitz, cherishing their time and accepting it as their new home.

How Does The Film Powerfully Present The Harrowing Events Of The Holocaust?

The most remarkable thing about The Zone of Interest is how it manages to say so much without directly saying it, combining the visual and the aural through a unique dissonance. With regards to the visuals, meaning scenes that play out to take forward the mostly simple and common story, the camera hardly ever leaves the confines of the host house. While some exceptions take place towards the latter part of the film, when Rudolf is transferred to a different concentration camp and he is seen at his new post, almost no scene of the camp in Auschwitz is seen. But the audio track picks up on numerous cries, lashes, and sounds that clearly come from the outside world but are heavily ignored. There is only one brief scene in which we are shown a side-angled close-up of Rudolf while he is at his workplace, which is a camp intended to kill Jews by the thousands. Indeed, the man is shot looking at the work he is rather proud of doing, amidst thick smoke bellowing out and loud cries and shrieks of helpless people. Rudolf certainly has no reservations about overseeing a genocide, but the film particularly shines with respect to how it uses the very usual to highlight the horrific context in the backdrop.

Early on in the film, Rudolfs family and his subordinates celebrate the mans birthday with a fancy cake, and all the Nazi soldiers come to his house to greet him. This merrymaking literally takes place all while hundreds, if not thousands, of families, are kept locked in the concentration camps and forced into the gas chambers. But nobody seems to notice, or rather, everyone pretends to look through the entire matter, as if nothing shocking is in the works. Rudolf is also seen meeting with a businessman in his house, who comes to show the commander plans and designs for a new, more effective gas chamber that he wants to build for his government. Rudolf goes through the plans without any hesitation and then also reports about this businessmans portfolio to his higher authorities, convinced that sturdier and better-designed gas chambers are needed to take his beloved nation and his government forward. The Zone of Interest does not really differentiate between evil-doers and those supporting such evil, but Rudolf is definitely in the first category, as he clearly enjoys the torture and killing of people.

What comes as even a bigger shock is the reaction of his wife, Hedwig, for she does not react to

any of these massacres either. Rather, the woman is extremely accustomed to the life of the commanders wife, and she enjoys the perks it brings along. She often receives luxury and expensive items that have been taken away from the prisoners, and on one particular occasion, she is seen receiving a fancy fur coat, since the Nazis did not differentiate between the rich and the poor among their targets. Hedwig immediately throws the coat on her body and tries it out in front of the mirror, only to realize that there is still lipstick lying inside one of the pockets. The presence of the lipstick would obviously be a bold reminder to anyone of the previous owner of the coat and the atrocious torture she must be subjected to at present. However, Hedwig has been wired to not think like that, and instead of any guilt or remorse, she feels rather excited to try on the lipstick, which is now hers as well.

Hedwig maintains a calm and composed nature, without any worry in the world, as she focuses on her gardening and getting a pool built for her children in their compound. The thick, dark smoke from the chimneys of the gas chambers on one side and from the steam engine train that brings in Jewish prisoners every day on the other does not affect the woman at all. The irony of the matter is all the more glaring when Hedwig is absolutely livid that her husband has to be transferred away from Auschwitz. She decides to stay back at the place along with her children because she is unwilling to uproot the life she had built there, including the fancy garden and the greenhouse, and shift somewhere else, which is probably too cold for her comfort as well. The fact that thousands were being faced with worse persecution and millions more would be uprooted, killed, or left disbanded very easily eludes her thought. In this regard, Jonathan Glazers film is a really fascinating note on not just the Holocaust but also the effect of systematized violence and the tendency of the masses to side with the oppressors in any given scenario.

The Hoss children are also equally desensitized to seeing murder and killing around them. The boys play around with toy soldiers, all waging war against invisible enemies. Shockingly, they are also seen collecting and playing with gold teeth, which are literally the remains of people who had been killed in the camp. One of the daughters does seem to feel something odd about their house, or she simply sleepwalks as a habit and sits by the door as if waiting for someone to arrive. Nonetheless, this young girl would also grow accustomed to the situation one day and not find anything strange about it. The Jewish prisoners are allowed to get close to the house and the family, as many of them are given the task of cleaning the boots and bringing supplies to the place. But there is also a clear distinction that the Nazi commander maintains from them, which highlights the pure hatred breeding

inside his perspective of the people. As soon as Rudolf finds a skull and some ashes in the river that he and his sons were bathing in, he scurries back to his house, and the children are scrubbed clean with utmost precision. In another instance, it is suggested that Rudolf forces himself upon a helpless prisoner woman, but he ensures that he scrubs his private parts before retiring for the night.

The only exception to the unaffected response by the entire family is by Hedwigs mother, who finds it bizarre that her daughter, her husband, and their children can really live at such a place. The elderly woman definitely has no sympathies for the prisoners, though, but she is rather unable to live with so many signs and reminders of death all around. The stench of burning human bodies and the ash flying around keep her up all night, but the very same elements are like playthings for the two young boys who still lay awake in their room. On a similar night, filled with reminders of the ongoing genocide, Hedwig is seen asking Rudolf to take her on a romantic trip, in the most romantic conversation between the couple in the film. Ultimately, the mother leaves the house unannounced very early the next morning, only leaving behind a note for her daughter. Hedwig simply tosses the note into her furnace insignificantly, almost insulted that someone would find her beloved home distasteful or discomforting. Even after Rudolf leaves for Oranienburg, Hedwig stays at their Auschwitz house with the children.

What Do The Scenes In Infrared Signify?

The Zone of Interest also sparsely presents a few scenes, in which an unacquainted young girl is seen going around Auschwitz, hiding apples and other meager food items inside the trenches. She is clearly doing this extremely dangerous work only to help the prisoners and ease their suffering in whatever little way she can. But interestingly, these scenes are in infrared, or negative, although only as long as the girl is in the outside world. As soon as she returns home, the visuals turn normal, then switch to infrared when she or her mother step out on the balcony. The family is revealed to be Polish locals who have no interest in Nazi ideals and dream of liberation one day. However, the mere fact that the family is still alive, irrespective of whether they are Jewish or not, suggests that they also have to work as collaborators for the Nazis to a certain degree. This was definitely the case with numerous non-Jews during the Nazi occupation who had to work for the horrific authorities despite not wanting to. Going by that logic, the significance of the use of infrared might be in stating how the family cannot be themselves as soon as they step out of their house or into the open balcony as well. Although the girl takes on the dangerous responsibility of helping the prisoners, she still cannot express her true self in public, leading to her being shown in infrared. Another

perspective is that the girl and her mother truly stand out in this horrific world solely because of their generous actions. Therefore, in a film like this, in which the Nazis and the enablers are the normal people, anyone with any sense of humanity has to be visually differentiated from the Hoss family members.

What Happens To Rudolf Hoss?

During The Zone of Interests ending, Rudolf is seen in his Berlin office as he telephones Hedwig and tells her about his excitement for the concentration camps being built. Rudolf had been given the responsibility of overseeing a new Nazi order in which Hungarian Jews were to be arrested and killed. Although Hedwig refuses to be part of this very direct talk of violence, for she prefers such matters to be in the background, the commander still feels thrilled. He is seen walking down the stairs from his office when suddenly bouts of violent retching hit him on two occasions. In the middle of these two instances, The Zone of Interest briefly moves to modern times, and various reminders of the Holocaust are seen being maintained at the Auschwitz-Birkenau State Museum, right before the place is opened to public visitors. The last scene returns to the past once more, and Rudolf is seen feeling slightly odd, as if someone is watching him, as he continues down the stairs.

The Zone of Interests ending scene seems to suggest that deep in his conscience, Rudolf Hoss does know that his actions can only make one retch, and almost like a fortune-teller, he has an uneasy feeling that his legacy will go down terribly in history. The scene of the museum is a fast jolt back to the right perspective, which had been missing throughout the film. Throughout the entire duration of The Zone of Interest, Rudolf, his family, and his professional associates had all been extremely invested in hiding the evidence and changing the narrative, but ultimately, the thousands of shoes or the torn, ragged uniforms still exist as reminders of the horrible genocide.

Document 10 (Source: https://www.loonyparty.com/about/policy-proposals/)

manicfesto proposals

26m tonnes of waste plastic bottles are discarded every year in the UK of which only 45% are recycled. The Loony Party has the answer.. Stop making them..

Before you askWe have found an alternative. Its called glass.

Some of our Proposals for other elections

Along with the existing Government policy for levelling up the North with the South we will

provide free Spirit Levels to all

We will reduce inflation by giving everyone free pins.

To make trains safer, we will fit them all with cushions on the front.

Any possible schemes thought up by Government, Council , NHS etc, such as closure of Hosptitals, workplace parking levy etc will be preceded with a Public Consultation which we will then ignore.

We will combat corruption in public life by taking part in it openly, we will introduce the Board of Bribery who will set standardised rates?. #sleaze for the many not just the few

We propose to prevent identity theft instantly by calling everyone Chris.

All political and electoral leaflets will be printed on soft paper so that it may be recycled in the appropriate manner.

In an effort to reduce the problems faced by the NHS , it is proposed to reduce pregnancy from nine to seven months ?

To protect pets and people of a nervous disposition we would introduce silent fireworks.

With Government helped finance, AstraZeneca should buy out Pfizer, then, as we would have the rights to Viagra, the economy may stay up longer.

Redundant Red Phones boxes will be converted to bijou accommodation to ease the housing shortage.

To make things fairer we will introduce a Court of Human Lefts.

General Election 2022 Manicfesto

General Election 2022 Manicfesto For the Manic, Not the Few

We pledge to fight this election on an invisible platform so that people cannot see the floors in our policies.

Once in Government, we will replace the Foreign Secretary with a British one!

Waitng Lists

We will reduce hospital waiting lists by using a smaller font.

Immigration

We will reduce net migration by making sure that any nets are secured more firmly to the ground.

Inflation

We will reduce inflation by giving everyone free pins

Government Policy

When formulating Policies the Government relies heavily on Expert Advise. Remember Experts built the Titanic

The Loony Party will also take into account the opinion of Dave on Facebook

Energy Policy

We will get rid of the Energy Price Cap and replace it with a Top Hat (This will also help our Millenery Industry)

2. We will get rid of all Standing Charges. (We are quite capable of sitting down and freezing to death)

3. All the hot air spoken in Parliament will be redirected to the Gas Distribution Networks.

Stressful times in the House

In order to calm down the passions and stresses currently exhibited in Parliament, the Loony Party

would make all M.Ps have half an hours compulsory Tai chi everyday.
This would counteract the other 23 ½ hours Chi Ting they do for the rest of the time

Corruption

We in The Loony Party are quite willing to accept bribes , and inducements from the Government in exchange that we dont stand in the election.
We will combat corruption in public life by taking part in it openly, we will also introduce the Board of Bribery who will set standardised rates?
Northern Powerhouse

The Loony party will invest millions in the Northern Powerhouse.
For clarification all parties agree that, as normal, the North starts at Hadrians Wall and ends where Scotland starts

Brexit

The Border in Northern Ireland would be made out of sponge to prevent a Hard Border
We will renegotiate to stay and lead the E.U and then sack the other 27 countries
Identity Theft

We propose to prevent identity theft instantly by calling everyone Dave.

Play Grounds

We will redevelop Playgrounds for all age groups.

Civil Service

The Civil Service will be extended to all branches of government, because a little politeness goes a long way.

Culture

The British Museum should have a Daddys section alongside the current Mummy exhibition.??

Transport

We will only paint yellow lines where you CAN park. Potholes deeper than 3 inches will be marked with a yellow plastic duck .

Elections

All political and electoral leaflets will be printed on soft Toilet paper so that it may be recycled in the appropriate manner. ??

NHS

In an effort to reduce the problems faced by the NHS , it is proposed to reduce pregnancy from nine to seven months ?

Animal Welfare

To protect pets and people of a nervous disposition we would introduce silent fireworks.?

General Election 2019 Manicfesto

General Election 2019 Manicfesto For the Manic, Not the Few

We pledge to fight this election on an invisible platform so that people cannot see the floors in our policies.

Stressful times in the House

In order to calm down the passions and stresses currently exhibited in Parliament, the Loony Party would make all M.Ps have half an hours compulsory Tai chi everyday.

This would counteract the other 23 ½ hours Chi Ting they do for the rest of the time

Corruption

We in The Loony Party are quite willing to accept bribes , and inducements from the Government in exchange that we dont stand in the election.
We will combat corruption in public life by taking part in it openly, we will also introduce the Board of Bribery who will set standardised rates?
Northern Powerhouse

The Loony party will invest millions in the Northern Powerhouse.
For clarification all parties agree that, as normal, the North starts at Hadrians Wall and ends where Scotland starts

Brexit

The Border in Northern Ireland would be made out of sponge to prevent a Hard Border
We will renegotiate to stay and lead the E.U and then sack the other 27 countries
Identity Theft

We propose to prevent identity theft instantly by calling everyone Dave.

Play Grounds

We will redevelop Playgrounds for all age groups.

Civil Service

The Civil Service will be extended to all branches of government, because a little politeness goes a long way.

Culture

The British Museum should have a Daddys section alongside the current Mummy exhibition.??

Transport

We will rename the current Oyster cards, Sardine Cards to better reflect the experience when travelling on public transport

2. We will only paint yellow lines where you CAN park. Potholes deeper than 3 inches will be marked with a yellow plastic duck .

Elections

All political and electoral leaflets will be printed on soft Toilet paper so that it may be recycled in the appropriate manner. ??

NHS

In an effort to reduce the problems faced by the NHS , it is proposed to reduce pregnancy from nine to seven months ?

Animal Welfare

To protect pets and people of a nervous disposition we would introduce silent fireworks.?

Policies

We encourage everyone, even current politicians, to submit ideas to our world famous #Manicfesto! The following are some of the most recent from our wonderful Twitter followers

Once in Government, anyone applying for 7 figure salary positions with the World Health Organisation or as Govt Health Advisors, will have to answer 15 correct questions on WHO wants to be a Millionaire.

In Brexit Trade Deals: Germany will be required to pay for treatment of Measles, and Spain will be required to pay for cases of Spanish Flu. The French will pay for all accidents resulting from kissing & broken letters & the Dutch will split all future expenses 50/50.

We will place in law measures to stop panic buying as COVID19 restrictions take hold. Shoppers will only be permitted to buy one panic per person.

It is evident that the 10pm pub curfew is not working , We propose that pubs ask people to leave in

alphabetical order.

Shamefully Lord Sutch has never been allowed to take his place in the House of Lords. Nor were Duke Ellington, Count Basie or Lord Rockingham We will end this discrimination against musicians.

To unite the population, we will surround the UK with a large cardboard box so people can be both in and/or out of the EU. This will be known as Schrodingers Brexit.

To get more children reading, fish and chips will once again be wrapped in newspaper.

Once in Government we will introduce the Ministry of Clarity. The role of this Ministry will ensure that only the clearest clarity is made clear and the unclear clarity is cleared out. We hope that our position on this is now clear to all.

In Government, we will complete a 5 year Parliament in only 4 years. This policy not only ensures a 20% saving for the public purse but also gives everyone in the UK a year off from listening to our politicians.

The MOT is an annual test to ensure that your car is roadworthy. We will introduce a ROT, an annual test to make sure all roads are car worthy.

And from 1st January 2021, passports will be issued in the colour of political voting. Tories will be Blue, Labour will be Red, Greens will be green. Official Loonies will have leopard spots, and Lib Dems will be invisible.

Chinners

Foreign Policy

We will Admit Shamima Begum back to the country only when she accepts Screaming Lord Sutch as her saviour.

Ministry of Info

We will create a New Ministry of Information. It shall consist of the former board of directors of Cambridge Analytica. They already know everything.

Brexit Proposals

We will Send Noel Edmonds to negotiate Brexit because he understands Deal or No Deal.

There will be no need for a backstop to the Brexit negotiations. Well have Alec Stewart as wicket-keeper.

James Wallace

Educational Funding

The Loony Party proposes that all Schools would have a Jumble sale or fete or other fundraising event at least twice per month to help raise funds for those little extras. . . such as Desks, Books, paper, pens , etc

R.U. Seerius

Pensions triple lock

In keeping with the Labour Partys latest bid to get one or two pensioners to vote for them they have brought out a new policy guaranteeing the Triple lock on pensions until 2025 if they get voted in.

The Loony party of course will go one better and buy a padlock, and as its now safer than a bank, new mattresses for all pensioners on less than 20p per week.

R.U. Seerius

Document 11 (Source: https://timdettmers.com/2023/01/30/which-gpu-for-deep-learning/)

Which GPU(s) to Get for Deep Learning: My Experience and Advice for Using GPUs in Deep Learning

2023-01-30 by Tim Dettmers 1,664 Comments

Deep learning is a field with intense computational requirements, and your choice of GPU will fundamentally determine your deep learning experience. But what features are important if you want to buy a new GPU? GPU RAM, cores, tensor cores, caches? How to make a cost-efficient choice? This blog post will delve into these questions, tackle common misconceptions, give you an intuitive understanding of how to think about GPUs, and will lend you advice, which will help you to make a choice that is right for you.

This blog post is designed to give you different levels of understanding of GPUs and the new Ampere series GPUs from NVIDIA. You have the choice: (1) If you are not interested in the details of how GPUs work, what makes a GPU fast compared to a CPU, and what is unique about the new NVIDIA RTX 40 Ampere series, you can skip right to the performance and performance per dollar charts and the recommendation section. The cost/performance numbers form the core of the blog post and the content surrounding it explains the details of what makes up GPU performance.

(2) If you worry about specific questions, I have answered and addressed the most common questions and misconceptions in the later part of the blog post.

(3) If you want to get an in-depth understanding of how GPUs, caches, and Tensor Cores work, the best is to read the blog post from start to finish. You might want to skip a section or two based on your understanding of the presented topics.

Contents hide

Overview

This blog post is structured in the following way. First, I will explain what makes a GPU fast. I will discuss CPUs vs GPUs, Tensor Cores, memory bandwidth, and the memory hierarchy of GPUs and how these relate to deep learning performance. These explanations might help you get a more intuitive sense of what to look for in a GPU. I discuss the unique features of the new NVIDIA RTX 40 Ampere GPU series that are worth considering if you buy a GPU. From there, I make GPU recommendations for different scenarios. After that follows a Q&A section of common questions posed to me in Twitter threads; in that section, I will also address common misconceptions and some miscellaneous issues, such as cloud vs desktop, cooling, AMD vs NVIDIA, and others.

How do GPUs work?

If you use GPUs frequently, it is useful to understand how they work. This knowledge will help you to undstand cases where are GPUs fast or slow. In turn, you might be able to understand better why you need a GPU in the first place and how other future hardware options might be able to compete. You can skip this section if you just want the useful performance numbers and arguments to help you decide which GPU to buy. The best high-level explanation for the question of how GPUs work is

my following Quora answer:

Read Tim Dettmers answer to Why are GPUs well-suited to deep learning? on Quora
This is a high-level explanation that explains quite well why GPUs are better than CPUs for deep learning. If we look at the details, we can understand what makes one GPU better than another.

The Most Important GPU Specs for Deep Learning Processing Speed
This section can help you build a more intuitive understanding of how to think about deep learning performance. This understanding will help you to evaluate future GPUs by yourself. This section is sorted by the importance of each component. Tensor Cores are most important, followed by memory bandwidth of a GPU, the cache hierachy, and only then FLOPS of a GPU.

Tensor Cores
Tensor Cores are tiny cores that perform very efficient matrix multiplication. Since the most expensive part of any deep neural network is matrix multiplication Tensor Cores are very useful. In fast, they are so powerful, that I do not recommend any GPUs that do not have Tensor Cores.

It is helpful to understand how they work to appreciate the importance of these computational units specialized for matrix multiplication. Here I will show you a simple example of A*B=C matrix multiplication, where all matrices have a size of 32×32, what a computational pattern looks like with and without Tensor Cores. This is a simplified example, and not the exact way how a high performing matrix multiplication kernel would be written, but it has all the basics. A CUDA programmer would take this as a first draft and then optimize it step-by-step with concepts like double buffering, register optimization, occupancy optimization, instruction-level parallelism, and many others, which I will not discuss at this point.

To understand this example fully, you have to understand the concepts of cycles. If a processor runs at 1GHz, it can do 10^9 cycles per second. Each cycle represents an opportunity for computation. However, most of the time, operations take longer than one cycle. Thus we essentially have a queue where the next operations needs to wait for the next operation to finish. This is also called the latency of the operation.

Here are some important latency cycle timings for operations. These times can change from GPU

generation to GPU generation. These numbers are for Ampere GPUs, which have relatively slow caches.

Global memory access (up to 80GB): ~380 cycles
L2 cache: ~200 cycles
L1 cache or Shared memory access (up to 128 kb per Streaming Multiprocessor): ~34 cycles
Fused multiplication and addition, a*b+c (FFMA): 4 cycles
Tensor Core matrix multiply: 1 cycle

Each operation is always performed by a pack of 32 threads. This pack is termed a warp of threads. Warps usually operate in a synchronous pattern threads within a warp have to wait for each other. All memory operations on the GPU are optimized for warps. For example, loading from global memory happens at a granularity of 32*4 bytes, exactly 32 floats, exactly one float for each thread in a warp. We can have up to 32 warps = 1024 threads in a streaming multiprocessor (SM), the GPU-equivalent of a CPU core. The resources of an SM are divided up among all active warps. This means that sometimes we want to run fewer warps to have more registers/shared memory/Tensor Core resources per warp.

For both of the following examples, we assume we have the same computational resources. For this small example of a 32×32 matrix multiply, we use 8 SMs (about 10% of an RTX 3090) and 8 warps per SM.

To understand how the cycle latencies play together with resources like threads per SM and shared memory per SM, we now look at examples of matrix multiplication. While the following example roughly follows the sequence of computational steps of matrix multiplication for both with and without Tensor Cores, please note that these are very simplified examples. Real cases of matrix multiplication involve much larger shared memory tiles and slightly different computational patterns.

Matrix multiplication without Tensor Cores

If we want to do an A*B=C matrix multiply, where each matrix is of size 32×32, then we want to load memory that we repeatedly access into shared memory because its latency is about five times lower (200 cycles vs 34 cycles). A memory block in shared memory is often referred to as a memory tile or just a tile. Loading two 32×32 floats into a shared memory tile can happen in parallel by using 2*32 warps. We have 8 SMs with 8 warps each, so due to parallelization, we only need to do a single

sequential load from global to shared memory, which takes 200 cycles.

To do the matrix multiplication, we now need to load a vector of 32 numbers from shared memory A and shared memory B and perform a fused multiply-and-accumulate (FFMA). Then store the outputs in registers C. We divide the work so that each SM does 8x dot products (32×32) to compute 8 outputs of C. Why this is exactly 8 (4 in older algorithms) is very technical. I recommend Scott Grays blog post on matrix multiplication to understand this. This means we have 8x shared memory accesses at the cost of 34 cycles each and 8 FFMA operations (32 in parallel), which cost 4 cycles each. In total, we thus have a cost of:

200 cycles (global memory) + 8*34 cycles (shared memory) + 8*4 cycles (FFMA) = 504 cycles

Lets look at the cycle cost of using Tensor Cores.

Matrix multiplication with Tensor Cores

With Tensor Cores, we can perform a 4×4 matrix multiplication in one cycle. To do that, we first need to get memory into the Tensor Core. Similarly to the above, we need to read from global memory (200 cycles) and store in shared memory. To do a 32×32 matrix multiply, we need to do 8×8=64 Tensor Cores operations. A single SM has 8 Tensor Cores. So with 8 SMs, we have 64 Tensor Cores just the number that we need! We can transfer the data from shared memory to the Tensor Cores with 1 memory transfers (34 cycles) and then do those 64 parallel Tensor Core operations (1 cycle). This means the total cost for Tensor Cores matrix multiplication, in this case, is:

200 cycles (global memory) + 34 cycles (shared memory) + 1 cycle (Tensor Core) = 235 cycles.

Thus we reduce the matrix multiplication cost significantly from 504 cycles to 235 cycles via Tensor Cores. In this simplified case, the Tensor Cores reduced the cost of both shared memory access and FFMA operations.

This example is simplified, for example, usually each thread needs to calculate which memory to read and write to as you transfer data from global memory to shared memory. With the new Hooper (H100) architectures we additionally have the Tensor Memory Accelerator (TMA) compute these indices in hardware and thus help each thread to focus on more computation rather than computing

indices.

Matrix multiplication with Tensor Cores and Asynchronous copies (RTX 30/RTX 40) and TMA (H100)

The RTX 30 Ampere and RTX 40 Ada series GPUs additionally have support to perform asynchronous transfers between global and shared memory. The H100 Hopper GPU extends this further by introducing the Tensor Memory Accelerator (TMA) unit. the TMA unit combines asynchronous copies and index calculation for read and writes simultaneously so each thread no longer needs to calculate which is the next element to read and each thread can focus on doing more matrix multiplication calculations. This looks as follows.

The TMA unit fetches memory from global to shared memory (200 cycles). Once the data arrives, the TMA unit fetches the next block of data asynchronously from global memory. While this is happening, the threads load data from shared memory and perform the matrix multiplication via the tensor core. Once the threads are finished they wait for the TMA unit to finish the next data transfer, and the sequence repeats.

As such, due to the asynchronous nature, the second global memory read by the TMA unit is already progressing as the threads process the current shared memory tile. This means, the second read takes only 200 34 1 = 165 cycles.

Since we do many reads, only the first memory access will be slow and all other memory accesses will be partially overlapped with the TMA unit. Thus on average, we reduce the time by 35 cycles.

165 cycles (wait for async copy to finish) + 34 cycles (shared memory) + 1 cycle (Tensor Core) = 200 cycles.

Which accelerates the matrix multiplication by another 15%.

From these examples, it becomes clear why the next attribute, memory bandwidth, is so crucial for Tensor-Core-equipped GPUs. Since global memory is the by far the largest cycle cost for matrix multiplication with Tensor Cores, we would even have faster GPUs if the global memory latency could be reduced. We can do this by either increasing the clock frequency of the memory (more

cycles per second, but also more heat and higher energy requirements) or by increasing the number of elements that can be transferred at any one time (bus width).

Memory Bandwidth

From the previous section, we have seen that Tensor Cores are very fast. So fast, in fact, that they are idle most of the time as they are waiting for memory to arrive from global memory. For example, during GPT-3-sized training, which uses huge matrices the larger, the better for Tensor Cores we have a Tensor Core TFLOPS utilization of about 45-65%, meaning that even for the large neural networks about 50% of the time, Tensor Cores are idle.

This means that when comparing two GPUs with Tensor Cores, one of the single best indicators for each GPUs performance is their memory bandwidth. For example, The A100 GPU has 1,555 GB/s memory bandwidth vs the 900 GB/s of the V100. As such, a basic estimate of speedup of an A100 vs V100 is 1555/900 = 1.73x.

L2 Cache / Shared Memory / L1 Cache / Registers

Since memory transfers to the Tensor Cores are the limiting factor in performance, we are looking for other GPU attributes that enable faster memory transfer to Tensor Cores. L2 cache, shared memory, L1 cache, and amount of registers used are all related. To understand how a memory hierarchy enables faster memory transfers, it helps to understand how matrix multiplication is performed on a GPU.

To perform matrix multiplication, we exploit the memory hierarchy of a GPU that goes from slow global memory, to faster L2 memory, to fast local shared memory, to lightning-fast registers. However, the faster the memory, the smaller it is.

While logically, L2 and L1 memory are the same, L2 cache is larger and thus the average physical distance that need to be traversed to retrieve a cache line is larger. You can see the L1 and L2 caches as organized warehouses where you want to retrieve an item. You know where the item is, but to go there takes on average much longer for the larger warehouse. This is the essential difference between L1 and L2 caches. Large = slow, small = fast.

For matrix multiplication we can use this hierarchical separate into smaller and smaller and thus

faster and faster chunks of memory to perform very fast matrix multiplications. For that, we need to chunk the big matrix multiplication into smaller sub-matrix multiplications. These chunks are called memory tiles, or often for short just tiles.

We perform matrix multiplication across these smaller tiles in local shared memory that is fast and close to the streaming multiprocessor (SM) the equivalent of a CPU core. With Tensor Cores, we go a step further: We take each tile and load a part of these tiles into Tensor Cores which is directly addressed by registers. A matrix memory tile in L2 cache is 3-5x faster than global GPU memory (GPU RAM), shared memory is ~7-10x faster than the global GPU memory, whereas the Tensor Cores registers are ~200x faster than the global GPU memory.

Having larger tiles means we can reuse more memory. I wrote about this in detail in my TPU vs GPU blog post. In fact, you can see TPUs as having very, very, large tiles for each Tensor Core. As such, TPUs can reuse much more memory with each transfer from global memory, which makes them a little bit more efficient at matrix multiplications than GPUs.

Each tile size is determined by how much memory we have per streaming multiprocessor (SM) and how much we L2 cache we have across all SMs. We have the following shared memory sizes on the following architectures:

Volta (Titan V): 128kb shared memory / 6 MB L2
Turing (RTX 20s series): 96 kb shared memory / 5.5 MB L2
Ampere (RTX 30s series): 128 kb shared memory / 6 MB L2
Ada (RTX 40s series): 128 kb shared memory / 72 MB L2
We see that Ada has a much larger L2 cache allowing for larger tile sizes, which reduces global memory access. For example, for BERT large during training, the input and weight matrix of any matrix multiplication fit neatly into the L2 cache of Ada (but not other Us). As such, data needs to be loaded from global memory only once and then data is available throught the L2 cache, making matrix multiplication about 1.5 2.0x faster for this architecture for Ada. For larger models the speedups are lower during training but certain sweetspots exist which may make certain models much faster. Inference, with a batch size larger than 8 can also benefit immensely from the larger L2 caches.

Estimating Ada / Hopper Deep Learning Performance

This section is for those who want to understand the more technical details of how I derive the performance estimates for Ampere GPUs. If you do not care about these technical aspects, it is safe to skip this section.

Practical Ada / Hopper Speed Estimates

Suppose we have an estimate for one GPU of a GPU-architecture like Hopper, Ada, Ampere, Turing, or Volta. It is easy to extrapolate these results to other GPUs from the same architecture/series. Luckily, NVIDIA already benchmarked the A100 vs V100 vs H100 across a wide range of computer vision and natural language understanding tasks. Unfortunately, NVIDIA made sure that these numbers are not directly comparable by using different batch sizes and the number of GPUs whenever possible to favor results for the H100 GPU. So in a sense, the benchmark numbers are partially honest, partially marketing numbers. In general, you could argue that using larger batch sizes is fair, as the H100/A100 GPU has more memory. Still, to compare GPU architectures, we should evaluate unbiased memory performance with the same batch size.

To get an unbiased estimate, we can scale the data center GPU results in two ways: (1) account for the differences in batch size, (2) account for the differences in using 1 vs 8 GPUs. We are lucky that we can find such an estimate for both biases in the data that NVIDIA provides.

Doubling the batch size increases throughput in terms of images/s (CNNs) by 13.6%. I benchmarked the same problem for transformers on my RTX Titan and found, surprisingly, the very same result: 13.5% it appears that this is a robust estimate.

As we parallelize networks across more and more GPUs, we lose performance due to some networking overhead. The A100 8x GPU system has better networking (NVLink 3.0) than the V100 8x GPU system (NVLink 2.0) this is another confounding factor. Looking directly at the data from NVIDIA, we can find that for CNNs, a system with 8x A100 has a 5% lower overhead than a system of 8x V100. This means if going from 1x A100 to 8x A100 gives you a speedup of, say, 7.00x, then going from 1x V100 to 8x V100 only gives you a speedup of 6.67x. For transformers, the figure is 7%.

Using these figures, we can estimate the speedup for a few specific deep learning architectures

from the direct data that NVIDIA provides. The Tesla A100 offers the following speedup over the Tesla V100:

SE-ResNeXt101: 1.43x
Masked-R-CNN: 1.47x
Transformer (12 layer, Machine Translation, WMT14 en-de): 1.70x
Thus, the figures are a bit lower than the theoretical estimate for computer vision. This might be due to smaller tensor dimensions, overhead from operations that are needed to prepare the matrix multiplication like img2col or Fast Fourier Transform (FFT), or operations that cannot saturate the GPU (final layers are often relatively small). It could also be artifacts of the specific architectures (grouped convolution).

The practical transformer estimate is very close to the theoretical estimate. This is probably because algorithms for huge matrices are very straightforward. I will use these practical estimates to calculate the cost efficiency of GPUs.

Possible Biases in Estimates
The estimates above are for H100, A100 , and V100 GPUs. In the past, NVIDIA sneaked unannounced performance degradations into the gaming RTX GPUs: (1) Decreased Tensor Core utilization, (2) gaming fans for cooling, (3) disabled peer-to-peer GPU transfers. It might be possible that there are unannounced performance degradations in the RTX 40 series compared to the full Hopper H100.

As of now, one of these degradations was found for Ampere GPUs: Tensor Core performance was decreased so that RTX 30 series GPUs are not as good as Quadro cards for deep learning purposes. This was also done for the RTX 20 series, so it is nothing new, but this time it was also done for the Titan equivalent card, the RTX 3090. The RTX Titan did not have performance degradation enabled.

Currently, no degradation for Ada GPUs are known, but I update this post with news on this and let my followers on twitter know.

Advantages and Problems for RTX40 and RTX 30 Series

The new NVIDIA Ampere RTX 30 series has additional benefits over the NVIDIA Turing RTX 20 series, such as sparse network training and inference. Other features, such as the new data types, should be seen more as an ease-of-use-feature as they provide the same performance boost as Turing does but without any extra programming required.

The Ada RTX 40 series has even further advances like 8-bit Float (FP8) tensor cores. The RTX 40 series also has similar power and temperature issues compared to the RTX 30. The issue of melting power connector cables in the RTX 40 can be easily prevented by connecting the power cable correctly.

Sparse Network Training

Ampere allows for fine-grained structure automatic sparse matrix multiplication at dense speeds. How does this work? Take a weight matrix and slice it into pieces of 4 elements. Now imagine 2 elements of these 4 to be zero. Figure 1 shows how this could look like.

Figure 1: Structure supported by the sparse matrix multiplication feature in Ampere GPUs. The figure is taken from Jeff Pool's GTC 2020 presentation on Accelerating Sparsity in the NVIDIA Ampere Architecture by the courtesy of NVIDIA.

Figure 1: Structure supported by the sparse matrix multiplication feature in Ampere GPUs. The figure is taken from Jeff Pools GTC 2020 presentation on Accelerating Sparsity in the NVIDIA Ampere Architecture by the courtesy of NVIDIA.

When you multiply this sparse weight matrix with some dense inputs, the sparse matrix tensor core feature in Ampere automatically compresses the sparse matrix to a dense representation that is half the size as can be seen in Figure 2. After this compression, the densely compressed matrix tile is fed into the tensor core which computes a matrix multiplication of twice the usual size. This effectively yields a 2x speedup since the bandwidth requirements during matrix multiplication from shared memory are halved.

Figure 2: The sparse matrix is compressed to a dense representation before the matrix multiplication is performed.

Figure 2: The sparse matrix is compressed to a dense representation before the matrix multiplication is performed. The figure is taken from Jeff Pools GTC 2020 presentation on Accelerating Sparsity in the NVIDIA Ampere Architecture by the courtesy of NVIDIA.

I was working on sparse network training in my research and I also wrote a blog post about sparse training. One criticism of my work was that You reduce the FLOPS required for the network, but it does not yield speedups because GPUs cannot do fast sparse matrix multiplication. Well, with the addition of the sparse matrix multiplication feature for Tensor Cores, my algorithm, or other sparse training algorithms, now actually provide speedups of up to 2x during training.

Figure 3: The sparse training algorithm that I developed has three stages: (1) Determine the importance of each layer. (2) Remove the smallest, unimportant weights. (3) Grow new weights proportional to the importance of each layer. Read more about my work in my sparse training blog post.
Figure 3: The sparse training algorithm that I developed has three stages: (1) Determine the importance of each layer. (2) Remove the smallest, unimportant weights. (3) Grow new weights proportional to the importance of each layer. Read more about my work in my sparse training blog post.
While this feature is still experimental and training sparse networks are not commonplace yet, having this feature on your GPU means you are ready for the future of sparse training.

Low-precision Computation

In my work, Ive previously shown that new data types can improve stability during low-precision backpropagation.

Figure 4: Low-precision deep learning 8-bit datatypes that I developed. Deep learning training benefits from highly specialized data types. My dynamic tree datatype uses a dynamic bit that indicates the beginning of a binary bisection tree that quantized the range [0, 0.9] while all previous bits are used for the exponent. This allows to dynamically represent numbers that are both large and small with high precision.
Figure 4: Low-precision deep learning 8-bit datatypes that I developed. Deep learning training benefits from highly specialized data types. My dynamic tree datatype uses a dynamic bit that indicates the beginning of a binary bisection tree that quantized the range [0, 0.9] while all previous bits are used for the exponent. This allows to dynamically represent numbers that are both large and small with high precision.
Currently, if you want to have stable backpropagation with 16-bit floating-point numbers (FP16), the big problem is that ordinary FP16 data types only support numbers in the range [-65,504, 65,504]. If

your gradient slips past this range, your gradients explode into NaN values. To prevent this during FP16 training, we usually perform loss scaling where you multiply the loss by a small number before backpropagating to prevent this gradient explosion.

The BrainFloat 16 format (BF16) uses more bits for the exponent such that the range of possible numbers is the same as for FP32: [-3*10^38, 3*10^38]. BF16 has less precision, that is significant digits, but gradient precision is not that important for learning. So what BF16 does is that you no longer need to do any loss scaling or worry about the gradient blowing up quickly. As such, we should see an increase in training stability by using the BF16 format as a slight loss of precision.

What this means for you: With BF16 precision, training might be more stable than with FP16 precision while providing the same speedups. With 32-bit TensorFloat (TF32) precision, you get near FP32 stability while giving the speedups close to FP16. The good thing is, to use these data types, you can just replace FP32 with TF32 and FP16 with BF16 no code changes required!

Overall, though, these new data types can be seen as lazy data types in the sense that you could have gotten all the benefits with the old data types with some additional programming efforts (proper loss scaling, initialization, normalization, using Apex). As such, these data types do not provide speedups but rather improve ease of use of low precision for training.

Fan Designs and GPUs Temperature Issues

While the new fan design of the RTX 30 series performs very well to cool the GPU, different fan designs of non-founders edition GPUs might be more problematic. If your GPU heats up beyond 80C, it will throttle itself and slow down its computational speed / power. This overheating can happen in particular if you stack multiple GPUs next to each other. A solution to this is to use PCIe extenders to create space between GPUs.

Spreading GPUs with PCIe extenders is very effective for cooling, and other fellow PhD students at the University of Washington and I use this setup with great success. It does not look pretty, but it keeps your GPUs cool! This has been running with no problems at all for 4 years now. It can also help if you do not have enough space to fit all GPUs in the PCIe slots. For example, if you can find the space within a desktop computer case, it might be possible to buy standard 3-slot-width RTX 4090 and spread them with PCIe extenders within the case. With this, you might solve both the

space issue and cooling issue for a 4x RTX 4090 setup with a single simple solution.

Figure 5: 4x GPUs with PCIe extenders. It looks like a mess, but it is very effective for cooling. I used this rig for 2 years and cooling is excellent despite problematic RTX 2080 Ti Founders Edition GPUs.

Figure 5: 4x GPUs with PCIe extenders. It looks like a mess, but it is very effective for cooling. I used this rig for 4 years and cooling is excellent despite problematic RTX 2080 Ti Founders Edition GPUs.

3-slot Design and Power Issues

The RTX 3090 and RTX 4090 are 3-slot GPUs, so one will not be able to use it in a 4x setup with the default fan design from NVIDIA. This is kind of justified because it runs at over 350W TDP, and it will be difficult to cool in a multi-GPU 2-slot setting. The RTX 3080 is only slightly better at 320W TDP, and cooling a 4x RTX 3080 setup will also be very difficult.

It is also difficult to power a 4x 350W = 1400W or 4x 450W = 1800W system in the 4x RTX 3090 or 4x RTX 4090 case. Power supply units (PSUs) of 1600W are readily available, but having only 200W to power the CPU and motherboard can be too tight. The components maximum power is only used if the components are fully utilized, and in deep learning, the CPU is usually only under weak load. With that, a 1600W PSU might work quite well with a 4x RTX 3080 build, but for a 4x RTX 3090 build, it is better to look for high wattage PSUs (+1700W). Some of my followers have had great success with cryptomining PSUs have a look in the comment section for more info about that. Otherwise, it is important to note that not all outlets support PSUs above 1600W, especially in the US. This is the reason why in the US, there are currently few standard desktop PSUs above 1600W on the market. If you get a server or cryptomining PSUs, beware of the form factor make sure it fits into your computer case.

Power Limiting: An Elegant Solution to Solve the Power Problem?

It is possible to set a power limit on your GPUs. So you would be able to programmatically set the power limit of an RTX 3090 to 300W instead of their standard 350W. In a 4x GPU system, that is a saving of 200W, which might just be enough to build a 4x RTX 3090 system with a 1600W PSU feasible. It also helps to keep the GPUs cool. So setting a power limit can solve the two major problems of a 4x RTX 3080 or 4x RTX 3090 setups, cooling, and power, at the same time. For a 4x setup, you still need effective blower GPUs (and the standard design may prove adequate for this),

but this resolves the PSU problem.

Figure 6: Reducing the power limit has a slight cooling effect. Reducing the RTX 2080 Ti power limit by 50-60 W decreases temperatures slightly and fans run more silent.
Figure 6: Reducing the power limit has a slight cooling effect. Reducing the RTX 2080 Ti power limit by 50-60 W decreases temperatures slightly and fans run more silent.
You might ask, Doesnt this slow down the GPU? Yes, it does, but the question is by how much. I benchmarked the 4x RTX 2080 Ti system shown in Figure 5 under different power limits to test this. I benchmarked the time for 500 mini-batches for BERT Large during inference (excluding the softmax layer). I choose BERT Large inference since, from my experience, this is the deep learning model that stresses the GPU the most. As such, I would expect power limiting to have the most massive slowdown for this model. As such, the slowdowns reported here are probably close to the maximum slowdowns that you can expect. The results are shown in Figure 7.

Figure 7: Measured slowdown for a given power limit on an RTX 2080 Ti. Measurements taken are mean processing times for 500 mini-batches of BERT Large during inference (excluding softmax layer).
Figure 7: Measured slowdown for a given power limit on an RTX 2080 Ti. Measurements taken are mean processing times for 500 mini-batches of BERT Large during inference (excluding softmax layer).
As we can see, setting the power limit does not seriously affect performance. Limiting the power by 50W more than enough to handle 4x RTX 3090 decreases performance by only 7%.

RTX 4090s and Melting Power Connectors: How to Prevent Problems

There was a misconception that RTX 4090 power cables melt because they were bent. However, it was found that only 0.1% of users had this problem and the problem occured due to user error. Here a video that shows that the main problem is that cables were not inserted correctly.

So using RTX 4090 cards is perfectly safe if you follow the following install instructions:

If you use an old cable or old GPU make sure the contacts are free of debri / dust.

Use the power connector and stick it into the socket until you hear a *click* this is the most important part.

Test for good fit by wiggling the power cable left to right. The cable should not move.
Check the contact with the socket visually, there should be no gap between cable and socket.

8-bit Float Support in H100 and RTX 40 series GPUs

The support of the 8-bit Float (FP8) is a huge advantage for the RTX 40 series and H100 GPUs. With 8-bit inputs it allows you to load the data for matrix multiplication twice as fast, you can store twice as much matrix elements in your caches which in the Ada and Hopper architecture are very large, and now with FP8 tensor cores you get 0.66 PFLOPS of compute for a RTX 4090 this is more FLOPS then the entirety of the worlds fastest supercomputer in year 2007. 4x RTX 4090 with FP8 compute rival the faster supercomputer in the world in year 2010 (deep learning started to work just in 2009).

The main problem with using 8-bit precision is that transformers can get very unstable with so few bits and crash during training or generate non-sense during inference. I have written a paper about the emergence of instabilities in large language models and I also written a more accessible blog post.

The main take-way is this: Using 8-bit instead of 16-bit makes things very unstable, but if you keep a couple of dimensions in high precision everything works just fine.

Main results from my work on 8-bit matrix multiplication for Large Language Models (LLMs). We can see that the best 8-bit baseline fails to deliver good zero-shot performance. The method that I developed, LLM.int8(), can perform Int8 matrix multiplication with the same results as the 16-bit baseline.

But Int8 was already supported by the RTX 30 / A100 / Ampere generation GPUs, why is FP8 in the RTX 40 another big upgrade? The FP8 data type is much more stable than the Int8 data type and its easy to use it in functions like layer norm or non-linear functions, which are difficult to do with Integer data types. This will make it very straightforward to use it in training and inference. I think this will make FP8 training and inference relatively common in a couple of months.

If you want to read more about the advantages of Float vs Integer data types you can read my recent paper about k-bit inference scaling laws. Below you can see one relevant main result for Float vs Integer data types from this paper. We can see that bit-by-bit, the FP4 data type preserve

more information than Int4 data type and thus improves the mean LLM zeroshot accuracy across 4 tasks.

4-bit Inference scaling laws for Pythia Large Language Models for different data types. We see that bit-by-bit, 4-bit float data types have better zeroshot accuracy compared to the Int4 data types.

Raw Performance Ranking of GPUs

Below we see a chart of raw relevative performance across all GPUs. We see that there is a gigantic gap in 8-bit performance of H100 GPUs and old cards that are optimized for 16-bit performance.

Shown is raw relative transformer performance of GPUs. For example, an RTX 4090 has about 0.33x performance of a H100 SMX for 8-bit inference. In other words, a H100 SMX is three times faster for 8-bit inference compared to a RTX 4090.

For this data, I did not model 8-bit compute for older GPUs. I did so, because 8-bit Inference and training are much more effective on Ada/Hopper GPUs because of the 8-bit Float data type and Tensor Memory Accelerator (TMA) which saves the overhead of computing read/write indices which is particularly helpful for 8-bit matrix multiplication. Ada/Hopper also have FP8 support, which makes in particular 8-bit training much more effective.

I did not model numbers for 8-bit training because to model that I need to know the latency of L1 and L2 caches on Hopper/Ada GPUs, and they are unknown and I do not have access to such GPUs. On Hopper/Ada, 8-bit training performance can well be 3-4x of 16-bit training performance if the caches are as fast as rumored.

But even with the new FP8 tensor cores there are some additional issues which are difficult to take into account when modeling GPU performance. For example, FP8 tensor cores do not support transposed matrix multiplication which means backpropagation needs either a separate transpose before multiplication or one needs to hold two sets of weights one transposed and one non-transposed in memory. I used two sets of weight when I experimented with Int8 training in my LLM.int8() project and this reduced the overall speedups quite significantly. I think one can do better with the right algorithms/software, but this shows that missing features like a transposed matrix multiplication for tensor cores can affect performance.

For old GPUs, Int8 inference performance is close to the 16-bit inference performance for models below 13B parameters. Int8 performance on old GPUs is only relevant if you have relatively large models with 175B parameters or more. If you are interested in 8-bit performance of older GPUs, you can read the Appendix D of my LLM.int8() paper where I benchmark Int8 performance.

GPU Deep Learning Performance per Dollar

Below we see the chart for the performance per US dollar for all GPUs sorted by 8-bit inference performance. How to use the chart to find a suitable GPU for you is as follows:

Determine the amount of GPU memory that you need (rough heuristic: at least 12 GB for image generation; at least 24 GB for work with transformers)

While 8-bit inference and training is experimental, it will become standard within 6 months. You might need to do some extra difficult coding to work with 8-bit in the meantime. Is that OK for you? If not, select for 16-bit performance.

Using the metric determined in (2), find the GPU with the highest relative performance/dollar that has the amount of memory you need.

We can see that the RTX 4070 Ti is most cost-effective for 8-bit and 16-bit inference while the RTX 3080 remains most cost-effective for 16-bit training. While these GPUs are most cost-effective, they are not necessarily recommended as they do not have sufficient memory for many use-cases. However, it might be the ideal cards to get started on your deep learning journey. Some of these GPUs are excellent for Kaggle competition where one can often rely on smaller models. Since to do well in Kaggle competitions the method of how you work is more important than the models size, many of these smaller GPUs are excellent for Kaggle competitions.

The best GPUs for academic and startup servers seem to be A6000 Ada GPUs (not to be confused with A6000 Turing). The H100 SXM GPU is also very cost effective and has high memory and very strong performance. If I would build a small cluster for a company/academic lab, I would use 66-80% A6000 GPUs and 20-33% H100 SXM GPUs. If I get a good deal on L40 GPUs, I would also pick them instead of A6000, so you can always ask for a quote on these.

Shown is relative performance per US Dollar of GPUs normalized by the cost for a desktop

computer and the average Amazon and eBay price for each GPU. Additionally, the electricity cost of ownership for 5 years is added with an electricity price of 0.175 USD per kWh and a 15% GPU utilization rate. The electricity cost for a RTX 4090 is about $100 per year. How to read and interpret the chart: a desktop computer with RTX 4070 Ti cards owned for 5 years yields about 2x more 8-bit inference performance per dollar compared to a RTX 3090 GPU.

GPU Recommendations

I have a create a recommendation flow-chart that you can see below (click here for interactive app from Nan Xiao). While this chart will help you in 80% of cases, it might not quite work for you because the options might be too expensive. In that case, try to look at the benchmarks above and pick the most cost effective GPU that still has enough GPU memory for your use-case. You can estimate the GPU memory needed by running your problem in the vast.ai or Lambda Cloud for a while so you know what you need. The vast.ai or Lambda Cloud might also work well if you only need a GPU very sporadically (every couple of days for a few hours) and you do not need to download and process large dataset to get started. However, cloud GPUs are usually not a good option if you use your GPU for many months with a high usage rate each day (12 hours each day). You can use the example in the When is it better to use the cloud vs a dedicated GPU desktop/server? section below to determine if cloud GPUs are good for you.

GPU recommendation chart for Ada/Hopper GPUs. Follow the answers to the Yes/No questions to find the GPU that is most suitable for you. While this chart works well in about 80% of cases, you might end up with a GPU that is too expensive. Use the cost/performance charts above to make a selection instead. [interactive app]

Is it better to wait for future GPUs for an upgrade? The future of GPUs.

To understand if it makes sense to skip this generation and buy the next generation of GPUs, it makes sense to talk a bit about what improvements in the future will look like.

In the past it was possible to shrink the size of transistors to improve speed of a processor. This is coming to an end now. For example, while shrinking SRAM increased its speed (smaller distance, faster memory access), this is no longer the case. Current improvements in SRAM do not improve its performance anymore and might even be negative. While logic such as Tensor Cores get smaller, this does not necessarily make GPU faster since the main problem for matrix multiplication is to get memory to the tensor cores which is dictated by SRAM and GPU RAM speed and size.

GPU RAM still increases in speed if we stack memory modules into high-bandwidth modules (HBM3+), but these are too expensive to manufacture for consumer applications. The main way to improve raw speed of GPUs is to use more power and more cooling as we have seen in the RTX 30s and 40s series. But this cannot go on for much longer.

Chiplets such as used by AMD CPUs are another straightforward way forward. AMD beat Intel by developing CPU chiplets. Chiplets are small chips that are fused together with a high speed on-chip network. You can think about them as two GPUs that are so physically close together that you can almost consider them a single big GPU. They are cheaper to manufacture, but more difficult to combine into one big chip. So you need know-how and fast connectivity between chiplets. AMD has a lot of experience with chiplet design. AMDs next generation GPUs are going to be chiplet designs, while NVIDIA currently has no public plans for such designs. This may mean that the next generation of AMD GPUs might be better in terms of cost/performance compared to NVIDIA GPUs.

However, the main performance boost for GPUs is currently specialized logic. For example, the asynchronous copy hardware units on the Ampere generation (RTX 30 / A100 / RTX 40) or the extension, the Tensor Memory Accelerator (TMA), both reduce the overhead of copying memory from the slow global memory to fast shared memory (caches) through specialized hardware and so each thread can do more computation. The TMA also reduces overhead by performing automatic calculations of read/write indices which is particularly important for 8-bit computation where one has double the elements for the same amount of memory compared to 16-bit computation. So specialized hardware logic can accelerate matrix multiplication further.
Low-bit precision is another straightforward way forward for a couple of years. We will see widespread adoption of 8-bit inference and training in the next months. We will see widespread 4-bit inference in the next year. Currently, the technology for 4-bit training does not exists, but research looks promising and I expect the first high performance FP4 Large Language Model (LLM) with competitive predictive performance to be trained in 1-2 years time.

Going to 2-bit precision for training currently looks pretty impossible, but it is a much easier problem than shrinking transistors further. So progress in hardware mostly depends on software and algorithms that make it possible to use specialized features offered by the hardware.

We will probably be able to still improve the combination of algorithms + hardware to the year 2032,

but after that will hit the end of GPU improvements (similar to smartphones). The wave of performance improvements after 2032 will come from better networking algorithms and mass hardware. It is uncertain if consumer GPUs will be relevant at this point. It might be that you need an RTX 9090 to run run Super HyperStableDiffusion Ultra Plus 9000 Extra or OpenChatGPT 5.0, but it might also be that some company will offer a high-quality API that is cheaper than the electricity cost for a RTX 9090 and you want to use a laptop + API for image generation and other tasks.

Overall, I think investing into a 8-bit capable GPU will be a very solid investment for the next 9 years. Improvements at 4-bit and 2-bit are likely small and other features like Sort Cores would only become relevant once sparse matrix multiplication can be leveraged well. We will probably see some kind of other advancement in 2-3 years which will make it into the next GPU 4 years from now, but we are running out of steam if we keep relying on matrix multiplication. This makes investments into new GPUs last longer.

Question & Answers & Misconceptions

Do I need PCIe 4.0 or PCIe 5.0?

Generally, no. PCIe 5.0 or 4.0 is great if you have a GPU cluster. It is okay if you have an 8x GPU machine, but otherwise, it does not yield many benefits. It allows better parallelization and a bit faster data transfer. Data transfers are not a bottleneck in any application. In computer vision, in the data transfer pipeline, the data storage can be a bottleneck, but not the PCIe transfer from CPU to GPU. So there is no real reason to get a PCIe 5.0 or 4.0 setup for most people. The benefits will be maybe 1-7% better parallelization in a 4 GPU setup.

Do I need 8x/16x PCIe lanes?

Same as with PCIe 4.0 generally, no. PCIe lanes are needed for parallelization and fast data transfers, which are seldom a bottleneck. Operating GPUs on 4x lanes is fine, especially if you only have 2 GPUs. For a 4 GPU setup, I would prefer 8x lanes per GPU, but running them at 4x lanes will probably only decrease performance by around 5-10% if you parallelize across all 4 GPUs.

How do I fit 4x RTX 4090 or 3090 if they take up 3 PCIe slots each?

You need to get one of the two-slot variants, or you can try to spread them out with PCIe extenders. Besides space, you should also immediately think about cooling and a suitable PSU.

PCIe extenders might also solve both space and cooling issues, but you need to make sure that you have enough space in your case to spread out the GPUs. Make sure your PCIe extenders are long enough!

How do I cool 4x RTX 3090 or 4x RTX 3080?
See the previous section.

Can I use multiple GPUs of different GPU types?
Yes, you can! But you cannot parallelize efficiently across GPUs of different types since you will often go at the speed of the slowest GPU (data and fully sharded parallelism). So different GPUs work just fine, but parallelization across those GPUs will be inefficient since the fastest GPU will wait for the slowest GPU to catch up to a synchronization point (usually gradient update).

What is NVLink, and is it useful?
Generally, NVLink is not useful. NVLink is a high speed interconnect between GPUs. It is useful if you have a GPU cluster with +128 GPUs. Otherwise, it yields almost no benefits over standard PCIe transfers.

I do not have enough money, even for the cheapest GPUs you recommend. What can I do?
Definitely buy used GPUs. You can buy a small cheap GPU for prototyping and testing and then roll out for full experiments to the cloud like vast.ai or Lambda Cloud. This can be cheap if you train/fine-tune/inference on large models only every now and then and spent more time protoyping on smaller models.

What is the carbon footprint of GPUs? How can I use GPUs without polluting the environment?
I built a carbon calculator for calculating your carbon footprint for academics (carbon from flights to conferences + GPU time). The calculator can also be used to calculate a pure GPU carbon footprint. You will find that GPUs produce much, much more carbon than international flights. As such, you should make sure you have a green source of energy if you do not want to have an astronomical carbon footprint. If no electricity provider in our area provides green energy, the best way is to buy carbon offsets. Many people are skeptical about carbon offsets. Do they work? Are they scams?

I believe skepticism just hurts in this case, because not doing anything would be more harmful than

risking the probability of getting scammed. If you worry about scams, just invest in a portfolio of offsets to minimize risk.

I worked on a project that produced carbon offsets about ten years ago. The carbon offsets were generated by burning leaking methane from mines in China. UN officials tracked the process, and they required clean digital data and physical inspections of the project site. In that case, the carbon offsets that were produced were highly reliable. I believe many other projects have similar quality standards.

What do I need to parallelize across two machines?
If you want to be on the safe side, you should get at least +50Gbits/s network cards to gain speedups if you want to parallelize across machines. I recommend having at least an EDR Infiniband setup, meaning a network card with at least 50 GBit/s bandwidth. Two EDR cards with cable are about $500 on eBay.

In some cases, you might be able to get away with 10 Gbit/s Ethernet, but this is usually only the case for special networks (certain convolutional networks) or if you use certain algorithms (Microsoft DeepSpeed).

Is the sparse matrix multiplication features suitable for sparse matrices in general?
It does not seem so. Since the granularity of the sparse matrix needs to have 2 zero-valued elements, every 4 elements, the sparse matrices need to be quite structured. It might be possible to adjust the algorithm slightly, which involves that you pool 4 values into a compressed representation of 2 values, but this also means that precise arbitrary sparse matrix multiplication is not possible with Ampere GPUs.

Do I need an Intel CPU to power a multi-GPU setup?
I do not recommend Intel CPUs unless you heavily use CPUs in Kaggle competitions (heavy linear algebra on the CPU). Even for Kaggle competitions AMD CPUs are still great, though. AMD CPUs are cheaper and better than Intel CPUs in general for deep learning. For a 4x GPU built, my go-to CPU would be a Threadripper. We built dozens of systems at our university with Threadrippers, and they all work great no complaints yet. For 8x GPU systems, I would usually go with CPUs that your vendor has experience with. CPU and PCIe/system reliability is more important in 8x systems than

straight performance or straight cost-effectiveness.

Does computer case design matter for cooling?
No. GPUs are usually perfectly cooled if there is at least a small gap between GPUs. Case design will give you 1-3 C better temperatures, space between GPUs will provide you with 10-30 C improvements. The bottom line, if you have space between GPUs, cooling does not matter. If you have no space between GPUs, you need the right cooler design (blower fan) or another solution (water cooling, PCIe extenders), but in either case, case design and case fans do not matter.

Will AMD GPUs + ROCm ever catch up with NVIDIA GPUs + CUDA?
Not in the next 1-2 years. It is a three-way problem: Tensor Cores, software, and community.

AMD GPUs are great in terms of pure silicon: Great FP16 performance, great memory bandwidth. However, their lack of Tensor Cores or the equivalent makes their deep learning performance poor compared to NVIDIA GPUs. Packed low-precision math does not cut it. Without this hardware feature, AMD GPUs will never be competitive. Rumors show that some data center card with Tensor Core equivalent is planned for 2020, but no new data emerged since then. Just having data center cards with a Tensor Core equivalent would also mean that few would be able to afford such AMD GPUs, which would give NVIDIA a competitive advantage.

Lets say AMD introduces a Tensor-Core-like-hardware feature in the future. Then many people would say, But there is no software that works for AMD GPUs! How am I supposed to use them? This is mostly a misconception. The AMD software via ROCm has come to a long way, and support via PyTorch is excellent. While I have not seen many experience reports for AMD GPUs + PyTorch, all the software features are integrated. It seems, if you pick any network, you will be just fine running it on AMD GPUs. So here AMD has come a long way, and this issue is more or less solved.

However, if you solve software and the lack of Tensor Cores, AMD still has a problem: the lack of community. If you have a problem with NVIDIA GPUs, you can Google the problem and find a solution. That builds a lot of trust in NVIDIA GPUs. You have the infrastructure that makes using NVIDIA GPUs easy (any deep learning framework works, any scientific problem is well supported). You have the hacks and tricks that make usage of NVIDIA GPUs a breeze (e.g., apex). You can find experts on NVIDIA GPUs and programming around every other corner while I knew much less AMD

GPU experts.

In the community aspect, AMD is a bit like Julia vs Python. Julia has a lot of potential, and many would say, and rightly so, that it is the superior programming language for scientific computing. Yet, Julia is barely used compared to Python. This is because the Python community is very strong. Numpy, SciPy, Pandas are powerful software packages that a large number of people congregate around. This is very similar to the NVIDIA vs AMD issue.

Thus, it is likely that AMD will not catch up until Tensor Core equivalent is introduced (1/2 to 1 year?) and a strong community is built around ROCm (2 years?). AMD will always snatch a part of the market share in specific subgroups (e.g., cryptocurrency mining, data centers). Still, in deep learning, NVIDIA will likely keep its monopoly for at least a couple more years.

When is it better to use the cloud vs a dedicated GPU desktop/server?
Rule-of-thumb: If you expect to do deep learning for longer than a year, it is cheaper to get a desktop GPU. Otherwise, cloud instances are preferable unless you have extensive cloud computing skills and want the benefits of scaling the number of GPUs up and down at will.

Numbers in the following paragraphs are going to change, but it serves as a scenario that helps you to understand the rough costs. You can use similar math to determine if cloud GPUs are the best solution for you.

For the exact point in time when a cloud GPU is more expensive than a desktop depends highly on the service that you are using, and it is best to do a little math on this yourself. Below I do an example calculation for an AWS V100 spot instance with 1x V100 and compare it to the price of a desktop with a single RTX 3090 (similar performance). The desktop with RTX 3090 costs $2,200 (2-GPU barebone + RTX 3090). Additionally, assuming you are in the US, there is an additional $0.12 per kWh for electricity. This compares to $2.14 per hour for the AWS on-demand instance.

At 15% utilization per year, the desktop uses:

(350 W (GPU) + 100 W (CPU))*0.15 (utilization) * 24 hours * 365 days = 591 kWh per year

So 591 kWh of electricity per year, that is an additional $71.

The break-even point for a desktop vs a cloud instance at 15% utilization (you use the cloud instance 15% of time during the day), would be about 300 days ($2,311 vs $2,270):

$2.14/h * 0.15 (utilization) * 24 hours * 300 days = $2,311

So if you expect to run deep learning models after 300 days, it is better to buy a desktop instead of using AWS on-demand instances.

You can do similar calculations for any cloud service to make the decision if you go for a cloud service or a desktop.

Common utilization rates are the following:

PhD student personal desktop: < 15%
PhD student slurm GPU cluster: > 35%
Company-wide slurm research cluster: > 60%
In general, utilization rates are lower for professions where thinking about cutting edge ideas is more important than developing practical products. Some areas have low utilization rates (interpretability research), while other areas have much higher rates (machine translation, language modeling). In general, the utilization of personal machines is almost always overestimated. Commonly, most personal systems have a utilization rate between 5-10%. This is why I would highly recommend slurm GPU clusters for research groups and companies instead of individual desktop GPU machines.

Version History
2023-01-30: Improved font and recommendation chart. Added 5 years cost of ownership electricity perf/USD chart. Updated Async copy and TMA functionality. Slight update to FP8 training. General improvements.
2023-01-16: Added Hopper and Ada GPUs. Added GPU recommendation chart. Added information about the TMA unit and L2 cache.
2020-09-20: Added discussion of using power limiting to run 4x RTX 3090 systems. Added older

GPUs to the performance and cost/performance charts. Added figures for sparse matrix multiplication.

2020-09-07: Added NVIDIA Ampere series GPUs. Included lots of good-to-know GPU details.

2019-04-03: Added RTX Titan and GTX 1660 Ti. Updated TPU section. Added startup hardware discussion.

2018-11-26: Added discussion of overheating issues of RTX cards.

2018-11-05: Added RTX 2070 and updated recommendations. Updated charts with hard performance data. Updated TPU section.

2018-08-21: Added RTX 2080 and RTX 2080 Ti; reworked performance analysis

2017-04-09: Added cost-efficiency analysis; updated recommendation with NVIDIA Titan Xp

2017-03-19: Cleaned up blog post; added GTX 1080 Ti

2016-07-23: Added Titan X Pascal and GTX 1060; updated recommendations

2016-06-25: Reworked multi-GPU section; removed simple neural network memory section as no longer relevant; expanded convolutional memory section; truncated AWS section due to not being efficient anymore; added my opinion about the Xeon Phi; added updates for the GTX 1000 series

2015-08-20: Added section for AWS GPU instances; added GTX 980 Ti to the comparison relation

2015-04-22: GTX 580 no longer recommended; added performance relationships between cards

2015-03-16: Updated GPU recommendations: GTX 970 and GTX 580

2015-02-23: Updated GPU recommendations and memory calculations

2014-09-28: Added emphasis for memory requirement of CNNs

Acknowledgments

I thank Suhail for making me aware of outdated prices on H100 GPUs, Gjorgji Kjosev for pointing out font issues, Anonymous for pointing out that the TMA unit does not exist on Ada GPUs, Scott Gray for pointing out that FP8 tensor cores have no transposed matrix multiplication, and reddit and HackerNews users for pointing out many other improvements.

For past updates of this blog post, I want to thank Mat Kelcey for helping me to debug and test custom code for the GTX 970; I want to thank Sander Dieleman for making me aware of the shortcomings of my GPU memory advice for convolutional nets; I want to thank Hannes Bretschneider for pointing out software dependency problems for the GTX 580; and I want to thank Oliver Griesel for pointing out notebook solutions for AWS instances. I want to thank Brad Nemire for providing me with an RTX Titan for benchmarking purposes. I want to thank Agrin Hilmkil, Ari Holtzman, Gabriel Ilharco, Nam Pho for their excellent feedback on the previous version of this blog

post.

Document 12 (Source: https://gleam.run/cheatsheets/gleam-for-python-users/)

Gleam for Python users

Hello productive pragmatic Pythonistas!

a soft wavey boundary between two sections of the website

Comments

Variables

Match operator

Variables type annotations

Functions

Exporting functions

Function type annotations

Referencing functions

Labelled arguments

Operators

Constants

Blocks

Data types

Strings

Tuples

Lists

Dicts

Flow control

Case

Try

Type aliases

Custom types

Records

Unions

Opaque custom types

Modules

Imports

Named imports

Unqualified imports

Comments

Python

In Python, comments are written with a # prefix.

```
# Hello, Joe!
```

A docstring (matching) that occurs as the first statement in a module, function, class, or method definition will become the __doc__ attribute of that object.

```
def a_function():
    """Return some important data."""
    pass
```

Gleam

In Gleam, comments are written with a // prefix.

```
// Hello, Joe!
```

Comments starting with /// are used to document the following statement. Comments starting with //// are used to document the current module.

```
//// This module is very important.

/// The answer to life, the universe, and everything.
const answer: Int = 42
```

Variables

You can reassign variables in both languages.

Python

```
size = 50
size = size + 100
size = 1
```

Python has no specific variable keyword. You choose a name and thats it!

Gleam

Gleam has the let keyword before its variable names.

```
let size = 50
let size = size + 100
let size = 1
```

Match operator

Python

Python supports basic, one directional destructuring (also called unpacking). Tuple of values can be unpacked and inner values can be assigned to left-hand variable names.

```
(a, b) = (1, 2)
# a == 1
# b == 2

# works also for for-loops
for key, value in enumerate(a_dict):
    print(key, value)
```

Gleam

In Gleam, let and = can be used for pattern matching, but youll get compile errors if theres a type mismatch, and a runtime error if theres a value mismatch. For assertions, the equivalent let assert keyword is preferred.

```
let #(x, _) = #(1, 2)
let assert [] = [1] // runtime error
let assert [y] = "Hello" // compile error, type mismatch
```

Variables type annotations

Python

Python is a dynamically typed language. Types are only checked at runtime and a variable can have different types in its lifetime.

Type hints (Python 3+) are optional annotations that document the code with type information. These annotations are accessible at runtime via the __annotations__ module-level variable.

These hints will mainly be used to inform static analysis tools like IDEs, linters

```
some_list: list[int] = [1, 2, 3]
```

Gleam

In Gleam type annotations can optionally be given when binding variables.

```
let some_list: List(Int) = [1, 2, 3]
```

Gleam will check the type annotation to ensure that it matches the type of the assigned value. It does not need annotations to type check your code, but you may find it useful to annotate variables to hint to the compiler that you want a specific type to be inferred.

Functions

Python

In Python, you can define functions with the def keyword. In that case, the return keyword is mandatory.

```
def sum(x, y):
    return x + y
```

Anonymous functions returning a single expression can also be defined with the lambda keyword and be assigned into variables.

```
mul = lambda x, y: x * y
mul(1, 2)
```

Gleam

Gleams functions are declared using a syntax similar to Rust or JavaScript. Gleams anonymous functions have a similar syntax and dont need a . when called.

```
pub fn sum(x, y) {
  x + y
}

let mul = fn(x, y) { x * y }
```

```
mul(1, 2)
```

Exporting functions

Python

In Python, top level functions are exported by default. There is no notion of private module-level functions.

Gleam

In Gleam, functions are private by default and need the pub keyword to be public.

```
// this is public
pub fn sum(x, y) {
  x + y
}

// this is private
fn mul(x, y) {
  x * y
}
```

Function type annotations

Python

Type hints can be used to optionally annotate function arguments and return types.

Discrepancies between type hints and actual values at runtime do not prevent interpretation of the code.

Static code analysers (IDE tooling, type checkers like mypy) will be required to detect those errors.

```
def sum(x: int, y: int) -> int:
    return x + y

def mul(x: int, y: int) -> bool:
    # no errors from the interpreter.
    return x * y
```

Gleam

Functions can optionally have their argument and return types annotated in Gleam. These type annotations will always be checked by the compiler and throw a compilation error if not valid. The compiler will still type check your program using type inference if annotations are omitted.

```
pub fn add(x: Int, y: Int) -> Int {
  x + y
}

pub fn mul(x: Int, y: Int) -> Bool { // compile error, type mismatch
  x * y
}
```

Referencing functions

Python

As long as functions are in scope they can be assigned to a new variable. There is no special syntax to assign a module function to a variable.

Gleam

Gleam has a single namespace for value and functions within a module, so there is no need for a special syntax to assign a module function to a variable.

```
fn identity(x) {
  x
}

fn main() {
  let func = identity
  func(100)
}
```

Labelled arguments

Both Python and Gleam have ways to give arguments names and in any order.

Python

Keyword arguments are evaluated once at function definition time, and there is no evidence showing a noticeable performance penalty when using named arguments.

When calling a function, arguments can be passed

positionally, in the same order of the function declaration
by name, in any order
def replace(inside: str, each: str, with_string: str):
    pass

# equivalent calls
replace('hello world', 'world', 'you')
replace(each='world', inside='hello world', with_string='you')
Gleam
In Gleam arguments can be given a label as well as an internal name. Contrary to Python, the name used at the call-site does not have to match the name used for the variable inside the function.

pub fn replace(inside string, each pattern, with replacement) {
  go(string, pattern, replacement)
}
replace(each: ",", with: " ", inside: "A,B,C")
There is no performance cost to Gleams labelled arguments as they are optimised to regular function calls at compile time, and all the arguments are fully type checked.

Operators
Operator Python Gleam Notes
Equal == == In Gleam both values must be of the same type
Strictly equal to == == Comparison in Gleam is always strict. (see note for Python)
Reference equality is True only if the two objects have the same reference
Not equal != != In Gleam both values must be of the same type
Greater than > > In Gleam both values must be ints
Greater than > >. In Gleam both values must be floats
Greater or equal >= >= In Gleam both values must be ints

Greater or equal >= >=. In Gleam both values must be floats

Less than < < In Gleam both values must be ints

Less than < <. In Gleam both values must be floats

Less or equal <= <= In Gleam both values must be ints

Less or equal <= <=. In Gleam both values must be floats

Boolean and and && In Gleam both values must be bools

Logical and and Not available in Gleam

Boolean or or || In Gleam both values must be bools

Logical or or Not available in Gleam

Add + + In Gleam both values must be ints

Add + +. In Gleam both values must be floats

Subtract - - In Gleam both values must be ints

Subtract - -. In Gleam both values must be floats

Multiply * * In Gleam both values must be ints

Multiply * *. In Gleam both values must be floats

Divide / / In Gleam both values must be ints

Divide / /. In Gleam both values must be floats

Remainder % % In Gleam both values must be ints, in Gleam negative values behave differently: Use int.modulo to mimick Pythons behavior.

Concatenate + <> In Gleam both values must be strings

Pipe |> Gleams pipe can pipe into anonymous functions. This operator does not exist in python

Some notes for Python:

== is by default comparing by value:

scalars will have their value compared

the only type cast will be for 0 and 1 that will be coerced to False and True respectively

variables that point to the same object will be equal with ==

two objects with the same members values wont be equal:

no structural equality, unless the __eq__ operator is redefined.

Python operators are short-circuiting as in Gleam.

Python operators can be overloaded and be applied to any types with potential custom behaviors

Constants

Python

In Python, top-level declarations are in the global/module scope is the highest possible scope. Any variables and functions defined will be accessible from anywhere in the code.

There is no notion of constant variables in Python.

```
# in the global scope
THE_ANSWER = 42
```

Gleam

In Gleam constants can be created using the const keyword.

```
const the_answer = 42

pub fn main() {
  the_answer
}
```

Blocks

Python

Python blocks are always associated with a function / conditional / class declarations There is no way to create multi-line expressions blocks like in Gleam.

Blocks are declared via indentation.

```
def a_func():
    # A block here
    pass
```

Gleam

In Gleam braces { } are used to group expressions.

```
pub fn main() {
  let x = {
    some_function(1)
```

```
  2
 }
 let y = x * {x + 10} // braces are used to change arithmetic operations order
 y
}
```

Data types

Strings

In Python, strings are stored as unicode code-points sequence. Strings can be encoded or decoded to/from a specific encoding.

In Gleam all strings are UTF-8 encoded binaries.

Python

```
"Hellø, world!"
```

Gleam

```
"Hellø, world!"
```

Tuples

Tuples are very useful in Gleam as theyre the only collection data type that allows mixed types in the collection.

Python

Python tuples are immutable, fixed-size lists that can contain mixed value types. Unpacking can be used to bind a name to a specific value of the tuple.

```
my_tuple = ("username", "password", 10)
_, password, _ = my_tuple
```

Gleam

```
let my_tuple = #("username", "password", 10)
let #(_, password, _) = my_tuple
```

Lists

Lists in Python are allowed to have values of mixed types, but not in Gleam.

Python

Python can emulate the cons operator of Gleam using the * operator and unpacking:

```
list = [2, 3, 4]
[head, *tail] = list
# head == 2
# tail == [3, 4]
```

Gleam

Gleam has a cons operator that works for lists destructuring and pattern matching. In Gleam lists are immutable so adding and removing elements from the start of a list is highly efficient.

```
let list = [2, 3, 4]
let list = [1, ..list]
let [1, second_element, ..] = list
[1.0, ..list] // compile error, type mismatch
```

Dictionaries

In Python, dictionaries can have keys of any type as long as:

the key type is hashable, such as integers, strings, tuples (due to their immutable values), functions and custom mutable objects implementing the __hash__ method.
the key is unique in the dictionary. and values of any type.

In Gleam, dicts can have keys and values of any type, but all keys must be of the same type in a given dict and all values must be of the same type in a given dict.

There is no dict literal syntax in Gleam, and you cannot pattern match on a dict. Dicts are generally not used much in Gleam, custom types are more common.

Python

```
{"key1": "value1", "key2": "value2"}
{"key1":  "1", "key2": 2}
```

Gleam

```
import gleam/dict

dict.from_list([#("key1", "value1"), #("key2", "value2")])
```

```
dict.from_list([#("key1", "value1"), #("key2", 2)]) // Type error!
```

Flow control

Case

Case is one of the most used control flows in Gleam. It can be seen as a switch statement on steroids. It provides a terse way to match a value type to an expression. Gleams case expression is fairly similar to Pythons match statement.

Python

Matching on primitive types:

```
def http_error(status):
    match status:
        case 400:
            return "Bad request"
        case 404:
            return "Not found"
        case 418:
            return "I'm a teapot"
```

Matching on tuples with variable capturing:

```
match point:
    case (0, 0):
        print("Origin")
    case (0, y):
        print(f"Y={y}")
    case (x, 0):
        print(f"X={x}")
    case (x, y):
        print(f""X={x}, Y={y}"")
    case _:
        raise ValueError("Not a point")
```

Matching on type constructors:

```
match point:
    case Point(x=0, y=0):
        print("Origin is the point's location.")
    case Point(x=0, y=y):
        print(f"Y={y} and the point is on the y-axis.")
    case Point(x=x, y=0):
        print(f"X={x} and the point is on the x-axis.")
    case Point():
        print("The point is located somewhere else on the plane.")
    case _:
        print("Not a point")
```

The match expression supports guards, similar to Gleam:

```
match point:
    case Point(x, y) if x == y:
        print(f"The point is located on the diagonal Y=X at {x}.")
    case Point(x, y):
        print(f"Point is not on the diagonal.")
```

Gleam

The case operator is a top level construct in Gleam:

```
case some_number {
  0 -> "Zero"
  1 -> "One"
  2 -> "Two"
  n -> "Some other number" // This matches anything
}
```

The case operator especially coupled with destructuring to provide native pattern matching:

```
case xs {
  [] -> "This list is empty"
  [a] -> "This list has 1 element"
  [a, b] -> "This list has 2 elements"
```

```
  _other -> "This list has more than 2 elements"
}
```

The case operator supports guards:

```
case xs {
  [a, b, c] if a >. b && a <=. c -> "ok"
  _other -> "ko"
}
```

and disjoint union matching:

```
case number {
  2 | 4 | 6 | 8 -> "This is an even number"
  1 | 3 | 5 | 7 -> "This is an odd number"
  _ -> "I'm not sure"
}
```

Try

Error management is approached differently in Python and Gleam.

Python

Python uses the notion of exceptions to interrupt the current code flow and pop up the error to the caller.

An exception is raised using the keyword raise.

```
def a_function_that_fails():
    raise Exception("an error")
```

The callee block will be able to capture any exception raised in the block using a try/except set of blocks:

```
try:
    print("executed")
    a_function_that_fails()
    print("not_executed")
```

```
except Exception as e:
    print("doing something with the exception", e)
```

Gleam

In contrast in Gleam, errors are just containers with an associated value.

A common container to model an operation result is Result(ReturnType, ErrorType).

A Result is either:

an Error(ErrorValue)

or an Ok(Data) record

Handling errors actually means to match the return value against those two scenarios, using a case for instance:

```
case int.parse("123") {
  Error(e) -> io.println("That wasn't an Int")
  Ok(i) -> io.println("We parsed the Int")
}
```

In order to simplify this construct, we can use the use expression with the try function from the gleam/result module.

bind a value to the providing name if Ok(Something) is matched

interrupt the flow and return Error(Something)

```
let a_number = "1"
let an_error = "ouch"
let another_number = "3"

use int_a_number <- try(parse_int(a_number))
use attempt_int <- try(parse_int(an_error)) // Error will be returned
use int_another_number <- try(parse_int(another_number)) // never gets executed

Ok(int_a_number + attempt_int + int_another_number) // never gets executed
```

Type aliases

Type aliases allow for easy referencing of arbitrary complex types. Even though their type systems does not serve the same function, both Python and Gleam provide this feature.

Python

A simple variable can store the result of a compound set of types.

```
type Headers = list[tuple[str, str]]

# can now be used to annotate a variable
headers: Headers = [("Content-Type", "application/json")]
```

Gleam

The type keyword can be used to create aliases:

```
pub type Headers =
  List(#(String, String))

let headers: Headers = [#("Content-Type", "application/json")]
```

Custom types

Records

Custom type allows you to define a collection data type with a fixed number of named fields, and the values in those fields can be of differing types.

Python

Python uses classes to define user-defined, record-like types. Properties are defined as class members and initial values are generally set in the constructor.

By default the constructor does not provide base initializers in the constructor so some boilerplate is needed:

```
class Person:
    name: str
    age: int
```

```
    def __init__(name: str, age: int) -> None:
        self.name = name
        self.age = age

person = Person(name="Jake", age=20)
# or with positional arguments Person("Jake", 20)
name = person.name
```

More recent alternatives are to use dataclasses or to leverage the NamedTuple base type to generate a constructor with initializers.

By default a class created with the dataclass decorator is mutable (although you can pass options to the dataclass decorator to change the behavior):

```
from dataclasses import dataclasses

@dataclass
class Person:
    name: str
    age: int

person = Person(name="Jake", age=20)
name = person.name
person.name = "John"  # The name is now "John"
```

NamedTuples on the other hand are immutable:

```
from typing import NamedTuple

class Person(NamedTuple):
    name: str
    age: int

person = Person(name="Jake", age=20)
```

```
name = person.name

# cannot reassign a value
person.name = "John"  # error
```

Gleam

Gleams custom types can be used in much the same way that structs are used in Elixir. At runtime, they have a tuple representation and are compatible with Erlang records.

```
type Person {
  Person(name: String, age: Int)
}

let person = Person(name: "Jake", age: 35)
let name = person.name
```

An important difference to note is there is no OOP in Gleam. Methods can not be added to types.

Unions

In Python unions can be declared with the | operator.

In Gleam functions must always take and receive one type. To have a union of two different types they must be wrapped in a new custom type.

Python

```
def int_or_float(x: int | float) -> str:
    if isinstance(x, int):
        return f"It's an integer: {x}"
    else:
        return f"It's a float: {x}"
```

Gleam

```
type IntOrFloat {
  AnInt(Int)
  AFloat(Float)
}
```

```
fn int_or_float(x) {
  case x {
    AnInt(1) -> "It's an integer: 1"
    AFloat(1.0) -> "It's a float: 1.0"
  }
}
```

Opaque custom types

In Python, constructors cannot be marked as private. Opaque types can be imperfectly emulated using a class method and some magic property that only updates via the class factory method.

In Gleam, custom types can be defined as being opaque, which causes the constructors for the custom type not to be exported from the module. Without any constructors to import other modules can only interact with opaque types using the intended API.

Python

```
class OnlyCreatable:

    __create_key = object()

    @classmethod
    def create(cls, value):
        return OnlyCreatable(cls.__create_key, value)

    def __init__(self, create_key, value):
        assert(create_key == OnlyCreatable.__create_key), \
            "OnlyCreatable objects must be created using OnlyCreatable.create"
        self.value = value
```

Gleam

```
pub opaque type Identifier {
  Identifier(Int)
}
```

```
pub fn get_id() {
  Identifier(100)
}
```

Modules

Python

There is no special syntax to define modules as files are modules in Python

Gleam

Gleams file is a module and named by the file name (and its directory path). Since there is no special syntax to create a module, there can be only one module in a file.

```
// in file foo.gleam
pub fn identity(x) {
  x
}
// in file main.gleam
import foo // if foo was in a folder called `lib` the import would be `lib/foo`
pub fn main() {
  foo.identity(1)
}
```

Imports

Python

```
# inside module src/nasa/moon_base.py
# imports module src/nasa/rocket_ship.py
from nasa import rocket_ship

def explore_space():
    rocket_ship.launch()
```

Gleam

Imports are relative to the root src folder.

Modules in the same directory will need to reference the entire path from src for the target module, even if the target module is in the same folder.

```
// inside module src/nasa/moon_base.gleam
// imports module src/nasa/rocket_ship.gleam
import nasa/rocket_ship

pub fn explore_space() {
  rocket_ship.launch()
}
```

Named imports

Python

```
import unix.cat as kitty
```

Gleam

```
import unix/cat as kitty
```

Unqualified imports

Python

```
from animal.cat import Cat, stroke

def main():
    kitty = Cat(name="Nubi")
    stroke(kitty)
```

Gleam

```
import animal/cat.{Cat, stroke}

pub fn main() {
  let kitty = Cat(name: "Nubi")
  stroke(kitty)
}
```

Document 13 (Source: https://towardsdatascience.com/gpt-from-scratch-with-mlx-acf2defda30e)

GPT from Scratch with MLX

Define and train GPT-2 on your MacBook

Pranav Jadhav

Towards Data Science

Pranav Jadhav

.

Follow

Published in

Towards Data Science

31 min read

.

2 days ago

Table of Contents

Preparing the data

Install mlx and run the following imports.

```
import mlx.core as mx
```

```
import mlx.nn as nn
import mlx.optimizers as optim
import mlx.utils as utils
import numpy as np
import math
```

The first step to training an LLM is collecting a large corpus of text data and then tokenizing it. Tokenization is the process of mapping text to integers, which can be fed into the LLM. Our training corpus for this model will be the works of Shakespeare concatenated into one file. This is roughly 1 million characters and looks like this:

```
First Citizen:
Before we proceed any further, hear me speak.

All:
Speak, speak.

First Citizen:
You are all resolved rather to die than to famish?

All:
Resolved. resolved.

First Citizen:
First, you know Caius Marcius is chief enemy to the people.
...
```

First, we read the file as a single long string into the text variable. Then we use the set() function to get all the unique characters in the text which will be our vocabulary. By printing vocab you can see all the characters in our vocabulary as one string, and we have a total of 65 characters which till be our tokens.

```
# Creating the vocabulary
with open('input.txt', 'r', encoding='utf-8') as f:
    text = f.read()
```

```
vocab = sorted(list(set(text)))
vocab_size = len(vocab)

print(''.join(vocab))
# !$&',-.3:;?ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz
print(vocab_size)
# 65
```

Production models will use tokenization algorithms like byte-pair encoding to generate a larger vocabulary of sub-word chunks. Since our focus today is on the architecture, we will continue with character-level tokenization. Next, we will map our vocabulary to integers known as token IDs. Then we can encode our text into tokens and decode them back to a string.

```
# Create mapping from vocab to integers
itos = {i:c for i,c in enumerate(vocab)} # int to string
stoi = {c:i for i,c in enumerate(vocab)} # string to int
encode = lambda x: [stoi[c] for c in x]
decode = lambda x: ''.join([itos[i] for i in x])

print(encode("hello world"))
# [46, 43, 50, 50, 53, 1, 61, 53, 56, 50, 42]
print(decode(encode("hello world")))
# hello world
```

We use theenumerate() function to iterate over all characters and their index in the vocabulary and create a dictionary itos which maps integers to characters and stoi which maps strings to integers. Then we use these mappings to create our encode and decode functions. Now we can encode the entire text and split training and validation data.

```
data = encode(text)
split = int(0.9 * len(data))
train_data = data[:split]
val_data = data[split:]
```

Currently, our training data is just a very long string of tokens. However, we are trying to train our model to predict the next token some given previous tokens. Therefore our dataset should be

comprised of examples where the input is some string of tokens and the label is the correct next token. We need to define a model parameter called context length which is the maximum number of tokens used to predict the next token. Our training examples will be the length of our context length.

Lets look at the first ctx_len+1 tokens.

```
ctx_len = 8
print(train_data[:ctx_len + 1])
# [18, 47, 56, 57, 58,  1, 15, 47, 58]
# x: [18, 47, 56, 57, 58,  1, 15, 47] | y: 58
```

This is one training example where the input is 18, 47, 56, 57, 58, 1, 15, 47 and the desired output is 58. This is 8 tokens of context. However, we also want to train the model to predict the next token given only 7, 6, 5 0 tokens as context which is needed during generation. Therefore we also consider the 8 sub examples packed into this example:

```
ctx_len = 8
print(train_data[:ctx_len + 1])
# [18, 47, 56, 57, 58,  1, 15, 47, 58]
# 8 sub examples
# [18] --> 47
# [18, 47] --> 56
# [18, 47, 56] --> 57
# [18, 47, 56, 57] --> 58
# [18, 47, 56, 57, 58] --> 1
# [18, 47, 56, 57, 58, 1] --> 15
# [18, 47, 56, 57, 58, 1, 15] --> 47
# [18, 47, 56, 57, 58, 1, 15, 47] --> 58
```

Notice that the labels are simply the inputs shifted left.

```
print("inputs: ", train_data[:ctx_len])
print("labels: ", train_data[1:ctx_len+1]) # labels = inputs indexed 1 higher
# inputs: [18, 47, 56, 57, 58,  1, 15, 47]
# labels: [47, 56, 57, 58,  1, 15, 47, 58]
```

At index 0 the input is 18 and the label is 47. At index 1 the input is everything before and including index 1 which is [18, 47] and the label is 56, etc. Now that we understand that the labels are simply the input sequence indexed one higher we can build our datasets.

```
# Creating training and validation datasets
ctx_len = 8
X_train = mx.array([train_data[i:i+ctx_len] for i in range(0, len(train_data) - ctx_len, ctx_len)])
y_train = mx.array([train_data[i+1:i+ctx_len+1] for i in range(0, len(train_data) - ctx_len, ctx_len)])
X_val = mx.array([val_data[i:i+ctx_len] for i in range(0, len(val_data) - ctx_len, ctx_len)])
y_val = mx.array([val_data[i+1:i+ctx_len+1] for i in range(0, len(val_data) - ctx_len, ctx_len)])
```

We loop through the data and take chunks of size ctx_len as the inputs (X) and then take the same chunks but at 1 higher index as the labels (y). Then we take these Python lists and create mlx array objects from them. The model internals will be written with mlx so we want our inputs to be mlx arrays.

One more thing. During training we dont want to feed the model one example at a time, we want to feed it multiple examples in parallel for efficiency. This group of examples is called our batch, and the number of examples in a group is our batch size. Thus we define a function to generate batches for training.

```
def get_batches(X, y, b_size, shuffle=True):
    if shuffle:
        ix = np.arange(X.shape[0])
        np.random.shuffle(ix)
        ix = mx.array(ix)
        X = X[ix]
        y = y[ix]
    for i in range(0, X.shape[0], b_size):
        input = X[i:i+b_size]
        label = y[i:i+b_size]
        yield input, label
```

If shuffle=True, we shuffle the data by indexing it with a randomly shuffled index. Then we loop through our dataset and return batch-size chunks from input and label datasets. These chunks are

known as mini-batches and are just stacked examples that we process in parallel. These mini-batches will be our input to the model during training.

Heres an example of a minibatch of 4 examples with context length 8.

A single minibatch (image by author)

This minibatch packs 32 next-token prediction problems. The model will predict the next token for each token in the input and the labels will be used to calculate the loss. Notice that the labels contain the next token for each index of the inputs.

Youll want to keep this picture in your mind because the shapes of these tensors will get hairy. For now, just remember that we will input a tensor of shape (batch_size, ctx_len) to the model.

Coding GPT-2

Lets look at the GPT-2 architecture to get an overview of what we are trying to implement.

GPT-2 Architecture (image by author)

Dont worry if this looks confusing. We will implement it step by step from bottom to top. Lets start by implementing the input embeddings.

Input Embeddings

The purpose of the input embedding layer is to map token IDs to vectors. Each token will be mapped to a vector which will be its representation as it is forwarded through the model. The vectors for each token will accumulate and exchange information as they pass through the model and eventually be used to predict the next token. These vectors are called embeddings.

The simplest way to map token IDs to vectors is through a lookup table. We create a matrix of size (vocab_size, n_emb) where each row is the embedding vector for the corresponding token. This matrix is known as the embedding weights.

Embedding Layer (image by author)

The diagram shows an example embedding layer of size (65, 6). This means there are 65 tokens in the vocabulary and each one will be represented by a length 6 embedding vector. The inputted sequence will be used to index the embedding weights to get the vector corresponding to each token. Remember the minibatches we input into the model? Originally the minibatch is size (batch_size, ctx_len). After passing through the embedding layer it is size (batch_size, ctx_len, n_emb). Instead of each token being a single integer, each token is now a vector of length n_emb.

Lets define the embedding layer in code now.

```
n_emb = 6 # You can add these hyperparams at the top of your file
class GPT(nn.Module):
    def __init__(self):
        super().__init__()
        self.wte = nn.Embedding(vocab_size, n_emb)
```

We will define a class to organize our implementation. We subclass nn.Module to take advantage of mlxs features. Then in the init function, we call the superclass constructor and initialize our token embedding layer called wte .

Positional Embeddings

Next up is the positional embeddings. The purpose of positional embeddings is to encode information about the position of each token in the sequence. This can be added to our input embeddings to get a complete representation of each token that contains information about the tokens position in the sequence.

```
class GPT(nn.Module):
    def __init__(self):
        super().__init__()
        self.wte = nn.Embedding(vocab_size, n_emb) # token embeddings
        self.wpe = nn.Embedding(ctx_len, n_emb) # position embeddings
```

The position embeddings work the same as token embeddings, except instead of having a row for each token we have a row for each possible position index. This means our embedding weights will be of shape (ctx_len, n_emb). Now we implement the __call__ function in our GPT class. This

function will contain the forward pass of the model.

```
# Tensor shapes commented
def __call__(self, x):
    B, T = x.shape # (B = batch_size, T = ctx_len)
    tok_emb = self.wte(x) # (B, T, n_emb)
    pos_emb = self.wpe(mx.arange(T)) # (T, n_emb)
    x = tok_emb + pos_emb # (B, T, n_emb)
```

First, we break out the dimensions of our input into variables B and T for easy handling. In sequence modeling contexts B and T are usually used as shorthand for batch and time dimensions. In this case, the time dimension of our sequence is the context length.

Next, we calculate token and position embeddings. Notice that for the position embeddings, our input is mx.arange(T) . This will output an array of consecutive integers from 0 to T-1 which is exactly what we want because those are the positions we want to embed. After passing that through the embedding layer we will have a tensor of shape (T, n_emb) because the embedding layer plucks out the n_emb length vector for each of the T positions. Note that even though pos_emb is not the same shape as tok_emb we can add the two because mlx will broadcast, or replicate pos_emb across the batch dimension to allow elementwise addition. Finally, we perform the addition to get the new representations of the tokens with positional information.

Self-Attention

So far the representation vectors for each token have been calculated independently. They have not had the opportunity to exchange any information. This is intuitively bad in language modeling because the meaning and usage of words depend on the surrounding context. Self-attention is how we incorporate information from previous tokens into a given token.

First, lets consider a naive approach. What if we simply represented each token as the average of its representation vector and the vectors of all the tokens before it? This achieves our goal of packing information from previous tokens into the representation for a given token. Heres what it would look like.

image by author

But self-attention doesnt involve writing a for-loop. The key insight is we can achieve this previous token averaging with matrix multiplication!

image by author

By multiplying our input sequence on the left by a special matrix we get the desired result. This matrix is known as the attention weights. Notice that each row of the attention weight matrix specificies how much of each other token goes into the representation for any given token. For example in row two, we have [0.5, 0.5, 0, 0]. This means that row two of the result will be 0.5*token1 + 0.5*token2 + 0*token3 + 0*token4 , or the average of token1 and token2. Note that the attention weights are a lower-triangular matrix (zeros in upper right entries). This ensures that future tokens will not be included in the representation of a given token. This ensures that tokens can only communicate with the previous tokens because during generation the model will only have access to previous tokens.

Lets look at how we can construct the attention weight matrix.

image by author

Notice that if we create an array of zeros with -inf in the upper right entries and then perform row-wise softmax we get the desired attention weights. A good exercise is to step through the softmax calculation for a row to see how this works. The takeaway is that we can take some array of size (ctx_len, ctx_len) and softmax each row to get attention weights that sum to one.

Now we can leave the realm of naive self-attention. Instead of simply averaging previous tokens, we use arbitrary weighted sums over previous tokens. Notice what happens when we do row-wise softmax of an arbitrary matrix.

image by author

We still get weights that sum to one on each row. During training, we can learn the numbers in the matrix on the left which will specify how much each token goes into the representation for another

token. This is how tokens pay attention to each other. But we still havent understood where this matrix on the left came from. These pre-softmax attention weights are calculated from the tokens themselves, but indirectly through three linear projections.

Keys, Queries, and Values

image by author

Each token in our sequence emits 3 new vectors. These vectors are called keys, queries, and values. We use the dot product of the query vector of one token and the key vector of another token to quantify the affinity those two tokens have. We want to calculate the pairwise affinities of each token with every other token, therefore we multiply the query vector (4x3) with the key vector transposed (3x4) to get the raw attention weights (4x4). Due to the way matrix multiplication works the (i,j) entry in the raw attention weights will be the query of token i dot the key of token j or the affinity between the two. Thus we have calculated interactions between every token. However, we dont want past tokens interacting with future tokens so we apply a mask of -inf to the upper right entries to ensure they will zero out after softmax. Then we perform row-wise softmax to get the final attention weights. Instead of multiplying these weights directly with the input, we multiply them with the value projection. This results in the new representations.

Now that we understand attention conceptually, lets implement it.

```
class Attention(nn.Module):
    def __init__(self, head_size):
        super().__init__()
        self.head_size = head_size
        self.k_proj = nn.Linear(n_emb, head_size, bias=False)
        self.q_proj = nn.Linear(n_emb, head_size, bias=False)
        self.v_proj = nn.Linear(n_emb, head_size, bias=False)
```

We start by defining the key, query, and value projection layers. Note that instead of going from n_emb to n_emb, we project from n_emb to head_size. This doesnt change anything, it just means the new representations calculated by attention will be dimension head_size.

```
class Attention(nn.Module):
    def __init__(self, head_size):
        super().__init__()
        self.head_size = head_size
        self.k_proj = nn.Linear(n_emb, head_size, bias=False)
        self.q_proj = nn.Linear(n_emb, head_size, bias=False)
        self.v_proj = nn.Linear(n_emb, head_size, bias=False)
    def __call__(self, x): # shapes commented
        B, T, C = x.shape # (batch_size, ctx_len, n_emb)
        K = self.k_proj(x) # (B, T, head_size)
        Q = self.q_proj(x) # (B, T, head_size)
        V = self.v_proj(x) # (B, T, head_size)
```

The forward pass begins by calculating the key, query, and value projections. We also break out the input shape into the variables B, T, and C for future convenience.

```
class Attention(nn.Module):
    def __init__(self, head_size):
        super().__init__()
        self.head_size = head_size
        self.k_proj = nn.Linear(n_emb, head_size, bias=False)
        self.q_proj = nn.Linear(n_emb, head_size, bias=False)
        self.v_proj = nn.Linear(n_emb, head_size, bias=False)
    def __call__(self, x):
        B, T, C = x.shape # (batch_size, ctx_len, n_emb)
        K = self.k_proj(x) # (B, T, head_size)
        Q = self.q_proj(x) # (B, T, head_size)
        V = self.v_proj(x) # (B, T, head_size)
        attn_weights = (Q @ K.transpose([0, 2, 1])) / math.sqrt(self.head_size)
        # attn_weights.shape = (B, T, T)
```

Next, we calculate the attention weights. We only want to transpose the last two dimensions of the key tensor, because the batch dimension is just there so we can forward multiple training examples in parallel. The mlx transpose function expects the new order of the dimensions as input, so we pass it [0, 2, 1] to transpose the last two dimensions. One more thing: we scale the attention weights by

the inverse square root of head_size. This is known as scaled attention and the purpose is to ensure that when Q and K are unit variance, attn_weights will be unit variance. If the variance of attn_weights is high, then the softmax will map these small and large values to 0 or 1which results in less complex representations.

The next step is to apply the mask to ensure we are doing causal language modeling i.e. ensuring tokens cannot attend to future tokens.

```
class Attention(nn.Module):
    def __init__(self, head_size):
        super().__init__()
        self.head_size = head_size
        self.k_proj = nn.Linear(n_emb, head_size, bias=False)
        self.q_proj = nn.Linear(n_emb, head_size, bias=False)
        self.v_proj = nn.Linear(n_emb, head_size, bias=False)
        indices = mx.arange(ctx_len)
        mask = indices[:, None] < indices[None] # broadcasting trick
        self._causal_mask = mask * -1e9
    def __call__(self, x):
        B, T, C = x.shape # (batch_size, ctx_len, n_emb)
        K = self.k_proj(x) # (B, T, head_size)
        Q = self.q_proj(x) # (B, T, head_size)
        V = self.v_proj(x) # (B, T, head_size)
        attn_weights = (Q @ K.transpose([0, 2, 1])) / math.sqrt(self.head_size)
        # attn_weights.shape = (B, T, T)
```

We create the mask with a clever broadcasting trick. Lets say our ctx_len=4 like in the diagrams above. First, we use mx.arange(4) to set the indices variable to [0, 1, 2, 3].

image by author

Then we can index like so indices[:, None] to generate a column vector with the values of indices. Similarly, we can get a row vector using indices[None]. Then when we do the < comparison, mlx broadcasts the vectors because they have mismatching shapes so they cant be compared

elementwise. Broadcasting means mlx will replicate the vectors along the lacking dimension. This results in an elementwise comparison of two (4, 4) matrices which makes sense. Side note: I recommend familiarizing yourself with the details of broadcasting by reading this, it comes up all the time when dealing with tensors.

After the elementwise comparison, we are left with the following tensor:

[[False, True, True, True],
[False, False, True, True],
[False, False, False, True],
[False, False, False, False]]
Multiplying this tensor by -1e9, we get:

[[-0e+00, -1e+09, -1e+09, -1e+09],
[-0e+00, -0e+00, -1e+09, -1e+09],
[-0e+00, -0e+00, -0e+00, -1e+09],
[-0e+00, -0e+00, -0e+00, -0e+00]]
Now we have an additive mask. We can add this matrix to our attention weights to make all the upper right entries very large negative numbers. This will cause them to be zeroed out after the softmax operation. Also, note that we add _ as a prefix to the attribute name _causal_mask which marks it as a private variable. This signals to mlx that it is not a parameter and should not be updated during training.

```
class Attention(nn.Module):
    def __init__(self, head_size):
        super().__init__()
        self.head_size = head_size
        self.k_proj = nn.Linear(n_emb, head_size, bias=False)
        self.q_proj = nn.Linear(n_emb, head_size, bias=False)
        self.v_proj = nn.Linear(n_emb, head_size, bias=False)
        indices = mx.arange(ctx_len)
        mask = indices[:, None] < indices[None] # broadcasting trick
        self._causal_mask = mask * -1e9
```

```
    def __call__(self, x):
        B, T, C = x.shape # (batch_size, ctx_len, n_emb)
        K = self.k_proj(x) # (B, T, head_size)
        Q = self.q_proj(x) # (B, T, head_size)
        V = self.v_proj(x) # (B, T, head_size)
        attn_weights = (Q @ K.transpose([0, 2, 1])) / math.sqrt(self.head_size)
        # attn_weights.shape = (B, T, T)
        attn_weights = attn_weights + self._causal_mask
        attn_weights = mx.softmax(attn_weights, axis=-1)
        o = (attn_weights @ V) # (B, T, head_size)
```

Now we can softmax row-wise to get the final attention weights and multiply these weights by the values to get our output. Note we pass axis=-1 to softmax which specifies that we want to softmax across the last dimension which are the rows.

The final step is output linear projection and dropout.

```
dropout = 0.1 # add this with hyperparams at top of file
class Attention(nn.Module):
    def __init__(self, head_size):
        super().__init__()
        self.head_size = head_size
        self.k_proj = nn.Linear(n_emb, head_size, bias=False)
        self.q_proj = nn.Linear(n_emb, head_size, bias=False)
        self.v_proj = nn.Linear(n_emb, head_size, bias=False)
        indices = mx.arange(ctx_len)
        mask = indices[:, None] < indices[None] # broadcasting trick
        self._causal_mask = mask * -1e9
        self.c_proj = nn.Linear(head_size, n_emb) # output projection
        self.resid_dropout = nn.Dropout(dropout)
    def __call__(self, x):
        B, T, C = x.shape # (batch_size, ctx_len, n_emb)
        K = self.k_proj(x) # (B, T, head_size)
        Q = self.q_proj(x) # (B, T, head_size)
```

```
        V = self.v_proj(x) # (B, T, head_size)
        attn_weights = (Q @ K.transpose([0, 2, 1])) / math.sqrt(self.head_size)
        # attn_weights.shape = (B, T, T)
        attn_weights = attn_weights + self._causal_mask
        attn_weights = mx.softmax(attn_weights, axis=-1)
        o = (attn_weights @ V) # (B, T, head_size)
        o = self.c_proj(self.resid_dropout(o))
        return o
```

We add two new layers, c_proj and resid_dropout which are the output projection and residual dropout. The output projection is to return the vectors to their original dimension n_emb. The dropout is added for regularization and training stability which is important as we start layering the transformer blocks to get a deep network. And thats it for implementing one attention head!

Multi-Head Attention

Instead of having just one attention head LLMs often use multiple attention heads in parallel and concatenate their outputs to create the final representation. For example, lets say we had one attention head with head_size=64 so the vector it produced for each token was 64 dimensional. We could achieve the same thing with 4 parallel attention heads each with head_size=16 by concatenating their outputs to produce a 16x4 = 64 dimensional output. Multi-head attention allows the model to learn more complex representations because each head learns different projections and attention weights.

```
n_heads = 4
class MultiHeadAttention(nn.Module): # naive implementation
    def __init__(self):
        super().__init__()
        self.heads = [Attention(head_size // n_heads) for _ in range(n_heads)]
    def __call__(self, x):
        return mx.concatenate([head(x) for head in self.heads], axis=-1)
```

The straightforward implementation is to create a list of n_heads attention heads where each one has size equal to our final head size divided by n_heads. Then we concatenate the output of each head over the last axis. However, this implementation is inefficient and does not take advantage of the speed of tensors. Lets implement multi-head attention with the power of tensors.

```
head_size = 64 # put at top of file
class MultiHeadAttention(nn.Module):
    def __init__(self):
        super().__init__()
        self.k_proj = nn.Linear(n_emb, head_size, bias=False)
        self.q_proj = nn.Linear(n_emb, head_size, bias=False)
        self.v_proj = nn.Linear(n_emb, head_size, bias=False)
        indices = mx.arange(ctx_len)
        mask = indices[:, None] < indices[None] # broadcasting trick
        self._causal_mask = mask * -1e9
        self.c_proj = nn.Linear(head_size, n_emb) # output projection
        self.resid_dropout = nn.Dropout(dropout)
    def __call__(self, x):
        B, T, C = x.shape # (batch_size, ctx_len, n_emb)
        K = self.k_proj(x) # (B, T, head_size)
        Q = self.q_proj(x) # (B, T, head_size)
        V = self.v_proj(x) # (B, T, head_size)
```

We start with our single-head attention implementation. The __init__() function has not changed. The forward pass begins as normal with the creation of the key, query, and value projections.

```
head_size = 64 # put at top of file
n_heads = 8 # put at top of file
class MultiHeadAttention(nn.Module):
    def __init__(self):
        super().__init__()
        self.k_proj = nn.Linear(n_emb, head_size, bias=False)
        self.q_proj = nn.Linear(n_emb, head_size, bias=False)
        self.v_proj = nn.Linear(n_emb, head_size, bias=False)
        indices = mx.arange(ctx_len)
        mask = indices[:, None] < indices[None] # broadcasting trick
        self._causal_mask = mask * -1e9
        self.c_proj = nn.Linear(head_size, n_emb) # output projection
```

```
        self.resid_dropout = nn.Dropout(dropout)
    def __call__(self, x):
        B, T, C = x.shape # (batch_size, ctx_len, n_emb)
        K = self.k_proj(x) # (B, T, head_size)
        Q = self.q_proj(x) # (B, T, head_size)
        V = self.v_proj(x) # (B, T, head_size)
        mha_shape = (B, T, n_heads, head_size//n_heads)
        K = mx.as_strided(K, (mha_shape)) # (B, T, n_heads, head_size//n_heads)
        Q = mx.as_strided(Q, (mha_shape)) # (B, T, n_heads, head_size//n_heads)
        V = mx.as_strided(V, (mha_shape)) # (B, T, n_heads, head_size//n_heads)
```

The next thing we need to do is introduce a new dimension for the number of heads n_heads . In the naive implementation, we had separate attention objects each with their own key, query, and value tensors but now we have them all in one tensor, therefore we need a dimension for the heads. We define the new shape we want in mha_shape . Then we use mx.as_strided() to reshape each tensor to have the head dimension. This function is equivalent to view from pytorch and tells mlx to treat this array as a different shape. But we still have a problem. Notice that we if try to multiply Q @ K_t (where K_t is K transposed over its last 2 dims) to compute attention weights as we did before, we will be multiplying the following shapes:

(B, T, n_heads, head_size//n_heads) @ (B, T, head_size//n_heads, n_heads)
Result shape: (B, T, n_heads, n_heads)
This would result in a tensor of shape (B, T, n_heads, n_heads) which is incorrect. With one head our attention weights were shape (B, T, T) which makes sense because it gives us the interaction between each pair of tokens. So now our shape should be the same but with a heads dimension: (B, n_heads, T, T) . We achieve this by transposing the dimensions of keys, queries, and values after we reshape them to make n_heads dimension 1 instead of 2.

```
head_size = 64 # put at top of file
n_heads = 8 # put at top of file
class MultiHeadAttention(nn.Module):
    def __init__(self):
        super().__init__()
        self.k_proj = nn.Linear(n_emb, head_size, bias=False)
```

```
        self.q_proj = nn.Linear(n_emb, head_size, bias=False)
        self.v_proj = nn.Linear(n_emb, head_size, bias=False)
        indices = mx.arange(ctx_len)
        mask = indices[:, None] < indices[None] # broadcasting trick
        self._causal_mask = mask * -1e9
        self.c_proj = nn.Linear(head_size, n_emb) # output projection
        self.attn_dropout = nn.Dropout(dropout)
        self.resid_dropout = nn.Dropout(dropout)
    def __call__(self, x):
        B, T, C = x.shape # (batch_size, ctx_len, n_emb)
        K = self.k_proj(x) # (B, T, head_size)
        Q = self.q_proj(x) # (B, T, head_size)
        V = self.v_proj(x) # (B, T, head_size)
        mha_shape = (B, T, n_heads, head_size//n_heads)
                K = mx.as_strided(K, (mha_shape)).transpose([0, 2, 1, 3]) # (B, n_heads, T, head_size//n_heads)
                Q = mx.as_strided(Q, (mha_shape)).transpose([0, 2, 1, 3]) # (B, n_heads, T, head_size//n_heads)
                V = mx.as_strided(V, (mha_shape)).transpose([0, 2, 1, 3]) # (B, n_heads, T, head_size//n_heads)
        attn_weights = (Q @ K.transpose([0, 1, 3, 2])) / math.sqrt(Q.shape[-1]) # (B, n_heads, T, T)
        attn_weights = attn_weights + self._causal_mask[:T, :T]
        attn_weights = mx.softmax(attn_weights, axis=-1)
        attn_weights = self.attn_dropout(attn_weights)
        o = (attn_weights @ V) # (B, n_heads, T, head_size//n_heads)
```

Now we can calculate the correction attention weights. Notice that we scale the attention weights by the size of an individual attention head rather than head_size which would be the size after concatenation. We also apply dropout to the attention weights.

Finally, we perform the concatenation and apply the output projection and dropout.

```
head_size = 64 # put at top of file
```

```
n_heads = 8 # put at top of file
class MultiHeadAttention(nn.Module):
    def __init__(self):
        super().__init__()
        self.k_proj = nn.Linear(n_emb, head_size, bias=False)
        self.q_proj = nn.Linear(n_emb, head_size, bias=False)
        self.v_proj = nn.Linear(n_emb, head_size, bias=False)
        indices = mx.arange(ctx_len)
        mask = indices[:, None] < indices[None] # broadcasting trick
        self._causal_mask = mask * -1e9
        self.c_proj = nn.Linear(head_size, n_emb) # output projection
        self.attn_dropout = nn.Dropout(dropout)
        self.resid_dropout = nn.Dropout(dropout)
    def __call__(self, x):
        B, T, C = x.shape # (batch_size, ctx_len, n_emb)
        K = self.k_proj(x) # (B, T, head_size)
        Q = self.q_proj(x) # (B, T, head_size)
        V = self.v_proj(x) # (B, T, head_size)
        mha_shape = (B, T, n_heads, head_size//n_heads)
                K = mx.as_strided(K, (mha_shape)).transpose([0, 2, 1, 3]) # (B, n_heads, T, head_size//n_heads)
                Q = mx.as_strided(Q, (mha_shape)).transpose([0, 2, 1, 3]) # (B, n_heads, T, head_size//n_heads)
                V = mx.as_strided(V, (mha_shape)).transpose([0, 2, 1, 3]) # (B, n_heads, T, head_size//n_heads)
        attn_weights = (Q @ K.transpose([0, 1, 3, 2])) / math.sqrt(Q.shape[-1]) # (B, n_heads, T, T)
        attn_weights = attn_weights + self._causal_mask[:T, :T]
        attn_weights = mx.softmax(attn_weights, axis=-1)
        attn_weights = self.attn_dropout(attn_weights)
        o = (attn_weights @ V) # (B, n_heads, T, head_size//n_heads)
        o = o.transpose([0, 2, 1, 3]).reshape((B, T, head_size)) # concat heads
        o = self.c_proj(self.resid_dropout(o))
        return o
```

Since we have everything in one tensor, we can do some shape manipulation to do the concatenation. First, we move n_heads back to the second to last dimension with the transpose function. Then we reshape back to the original size to undo the splitting into heads we performed earlier. This is the same as concatenating the final vectors from each head. And thats it for multi-head attention! Weve gotten through the most intense part of our implementation.

MLP

The next part of the architecture is the multilayer perception or MLP. This is a fancy way of saying 2 stacked linear layers. Theres not much to be said here, it is a standard neural network.

```
class MLP(nn.Module):
    def __init__(self):
        super().__init__()
        self.c_fc = nn.Linear(n_emb, 4 * n_emb)
        self.gelu = nn.GELU()
        self.c_proj = nn.Linear(4 * n_emb, n_emb)
        self.dropout = nn.Dropout(dropout)
    def __call__(self, x):
        x = self.gelu(self.c_fc(x))
        x = self.c_proj(x)
        x = self.dropout(x)
        return x
```

We take the input and project it to a higher dimension with c_fc . Then we apply gelu nonlinearity and project it back down to the embedding dimension with c_proj . Finally, we apply dropout and return. The purpose of the MLP is to allow for some computation after the vectors have communicated during attention. We will stack these communication layers (attention) and computation layers (mlp) into a block.

Block

A GPT block consists of attention followed by an MLP. These blocks will be repeated to make the architecture deep.

```
class Block(nn.Module):
```

```
    def __init__(self):
        super().__init__()
        self.mlp = MLP()
        self.mha = MultiHeadAttention()
    def __call__(self, x):
        x = self.mha(x)
        x = self.mlp(x)
        return x
```

Now, we need to add two more features to improve training stability. Lets take a look at the architecture diagram again.

Layernorms and Skip Connections

image by author

We still need to implement the components highlighted in red. The arrows are skip connections. Instead of the input being transformed directly, the effect of the attention and MLP layers is additive. Their result is added to the input instead of directly replacing it. This is good for the training stability of deep networks since in the backward pass, the operands of an addition operation will receive the same gradient as their sum. Gradients can thus flow backwards freely which prevents issues like vanishing/exploding gradients that plague deep networks. Layernorm also helps with training stability by ensuring activations are normally distributed. Here is the final implementation.

```
class Block(nn.Module):
    def __init__(self):
        super().__init__()
        self.mlp = MLP()
        self.mha = MultiHeadAttention()
        self.ln_1 = nn.LayerNorm(dims=n_emb)
        self.ln_2 = nn.LayerNorm(dims=n_emb)
    def __call__(self, x):
        x = x + self.mha(self.ln_1(x))
        x = x + self.mlp(self.ln_2(x))
        return x
```

Layernorm is applied before multi-head attention and MLP. The skip connections are added with x = x + ... making the operations additive.

Forward Pass

With the Block defined, we can finish the full GPT-2 forward pass.

```
n_layers = 3 # put at top of file
class GPT(nn.Module):
    def __init__(self):
        super().__init__()
        self.wte = nn.Embedding(vocab_size, n_emb) # token embeddings
        self.wpe = nn.Embedding(ctx_len, n_emb) # position embeddings
        self.blocks = nn.Sequential(
            *[Block() for _ in range(n_layers)],
        ) # transformer blocks
        self.ln_f = nn.LayerNorm(dims=n_emb) # final layernorm
        self.lm_head = nn.Linear(n_emb, vocab_size) # output projection
    # Tensor shapes commented
    def __call__(self, x):
        B, T = x.shape # (B = batch_size, T = ctx_len)
        tok_emb = self.wte(x) # (B, T, n_emb)
        pos_emb = self.wpe(mx.arange(T)) # (T, n_emb)
        x = tok_emb + pos_emb # (B, T, n_emb)
        x = self.blocks(x) # (B, T, n_emb)
        x = self.ln_f(x) # (B, T, b_emb)
        logits = self.lm_head(x) # (B, T, vocab_size)
        return logits
```

We create a container for the blocks using nn.Sequential which takes any input and passes it sequentially through the contained layers. Then we can apply all the blocks with self.blocks(x) . Finally, we apply a layer norm and then the lm_head. The lm_head or language modeling head is just a linear layer that maps from the embedding dimension to the vocab size. The model will output a vector containing some value for each word in our vocabulary, or the logits. We can softmax the logits to get a probability distribution over the vocabulary which we can sample from to get the next

token. We will also use the logits to calculate the loss during training. There are just two more things we need to implement before we begin training.

Sampling

We need to write a generate function to sample from the model once training is complete. The idea is that we start with some sequence of our choice, then we predict the next token and append this to our sequence. Then we feed the new sequence in and predict the next token again. This continues until we decide to stop.

```
# method of GPT class
def generate(self, max_new_tokens):
  ctx = mx.zeros((1, 1), dtype=mx.int32)
```

We prompt the model with a single token, zero. Zero is the newline character so it is a natural place to start the generation since we just want to see how Shakespeare-like our model can get. Note that we initialize the shape to (1, 1) to simulate a single batch with a sequence length of one.

```
# method of GPT class
def generate(self, max_new_tokens):
  ctx = mx.zeros((1, 1), dtype=mx.int32)
  for _ in range(max_new_tokens):
    logits = self(ctx[:, -ctx_len:]) # pass in last ctx_len characters
    logits = logits[:, -1, :] # get logits for the next token
    next_tok = mx.random.categorical(logits, num_samples=1)
    ctx = mx.concatenate((ctx, next_tok), axis=1)
return ctx
```

Then we get the logits for the next token by passing in the last ctx_len characters to the model. However, our model output is of shape (B, T, vocab_size) since it predicts the next token logits for each token in the input. We use all of that during training, but now we only want the logits for the last token because we can use this to sample a new token. Therefore we index the logits to get the last element in the first dimension which is the sequence dimension. Then we sample the next token using the mx.random.categorical() function which takes the logits and the number of samples we want as input. This function will softmax the logits to turn them into a probability distribution and then randomly sample a token according to the probabilities. Finally, we concatenate the new token to

the context and repeat the process max_new_tokens number of times.

Initialization

The last thing to do is handle weight initialization which is important for training dynamics.

```
# method of GPT
def _init_parameters(self):
    normal_init = nn.init.normal(mean=0.0, std=0.02)
    residual_init = nn.init.normal(mean=0.0, std=(0.02 / math.sqrt(2 * n_layers)))
```

First, we define two different nn.init.normal functions. The first one is for initializing all linear and embedding layers. The second one is for initializing linear layers that are specifically residual projections i.e. the last linear layer inside multi-head attention and MLP. The reason for this special initialization is that it checks accumulation along the residual path as model depth increases according to the GPT-2 paper [2].

In mlx we can change the parameters of the model using the mx.update() function. Checking the docs, it expects a complete or partial dictionary of the new model parameters. We can see what this dictionary looks like by printing out self.parameters() inside the GPT class.

```
{'wte': {'weight': array([[-0.025084, -0.0197523, -0.0341617, ..., -0.0979123, -0.0830218,
-0.0784692],
       [-0.00777913, -0.117002, -0.0310708, ..., 0.0128591, 0.122941, 0.000414443],
       [0.0240044, -0.0859084, 0.0253116, ..., 0.108967, 0.0767123, 0.0221565],
       ...,
       [0.050729, -0.04578, 0.0685943, ..., -0.0496998, -0.00350879, -0.00631825],
       [0.00518804, 0.0499818, 0.0330045, ..., 0.0300661, 0.0431054, 0.000958906],
              [-0.0323007, 0.0132046, 0.0208218, ..., -0.0785159, 0.00436121, -0.00726994]],
dtype=float32)}, 'wpe': {'weight': array([[0.000797923, -0.0396898, -0.029047, ..., -0.0132273,
0.00684483, -0.0067624],
       [-0.0247021, -0.0274349, 0.0310587, ..., -0.100099, 0.0301566, -0.0178732],
       [0.0929172, -0.0468649, 0.0101506, ..., -0.0341086, -0.0516283, 0.0447596],
       ...,
       [-0.0508172, 0.0892201, -0.00183612, ..., -0.00341944, 0.023437, 0.0296461],
```

[0.0105829, 0.0688093, 0.146744, ..., -0.0836337, 0.0206679, 0.0184166],

[-0.00578717, -0.0606196, -0.0917056, ..., -0.0641549, -0.0490424, 0.0998114]], dtype=float32)}, 'blocks': {'layers': [{'mlp': {'c_fc': {'weight': array([[0.0169199, 0.00264431, 0.0316978, ..., -0.0596867, -0.0153549, 0.0176386],

...

Its a nested dictionary containing each model weight as an mx.array. So to initialize the parameters of our model we need to build up a dictionary like this with our new params and pass them to self.update() . We can achieve this as follows:

```
# method of GPT
def _init_parameters(self):
    normal_init = nn.init.normal(mean=0.0, std=0.02)
    residual_init = nn.init.normal(mean=0.0, std=(0.02 / math.sqrt(2 * n_layers)))
    new_params = []
    for name, module in self.named_modules():
        if isinstance(module, nn.layers.linear.Linear):
            new_params.append((name + '.weight', normal_init(module.weight)))
        elif isinstance(module, nn.layers.embedding.Embedding):
            new_params.append((name + '.weight', normal_init(module.weight)
```

We maintain a list of tuples called new_params which will contain tuples of (parameter_name, new_value). Next, we loop through each nn.Module object in our model with self.named_modules() which returns tuples of (name, module). If we print out the module names within the loop we see that they look like this:

```
lm_head
blocks
blocks.layers.4
blocks.layers.3
blocks.layers.3.ln_2
blocks.layers.3.ln_1
blocks.layers.3.mha
blocks.layers.3.mha.resid_dropout
blocks.layers.3.mha.c_proj
```

blocks.layers.3.mha.attn_dropout

blocks.layers.3.mha.c_attn

...

blocks.layers.0.mlp.dropout

blocks.layers.0.mlp.c_proj

blocks.layers.0.mlp.gelu

blocks.layers.0.mlp.c_fc

wpe

wte

We use the isinstance() function to find the linear and embedding layers and then add them to our list. For example, say we are looping and reach blocks.layers.0.mlp.c_fc which is the first linear layer in the MLP. This would trigger the first if statement, and the tuple ("block.layers.0.mlp.c_fc.weight", [<normally initialized weight here>]) would be added to our list. We have to add .weight to the name because we specifically want to initialize the weight in this way, not the bias. Now we need to handle the residual projection initialization.

```
# method of GPT
def _init_parameters(self):
    normal_init = nn.init.normal(mean=0.0, std=0.02)
    residual_init = nn.init.normal(mean=0.0, std=(0.02 / math.sqrt(2 * n_layers)))
    new_params = []
    for name, module in self.named_modules():
        if isinstance(module, nn.layers.linear.Linear):
            if 'c_proj' in name: # residual projection
                new_params.append((name + '.weight', residual_init(module.weight)))
            else:
                new_params.append((name + '.weight', normal_init(module.weight)))
        elif isinstance(module, nn.layers.embedding.Embedding):
            new_params.append((name + '.weight', normal_init(module.weight)))
```

After checking if the module is a linear layer, we check if c_proj is in the name because thats how we named the residual projections. Then we can apply the special initialization. Finally, we need to initialize the biases to be zero.

```
# method of GPT
def _init_parameters(self):
    normal_init = nn.init.normal(mean=0.0, std=0.02)
    residual_init = nn.init.normal(mean=0.0, std=(0.02 / math.sqrt(2 * n_layers)))
    new_params = []
    for name, module in self.named_modules():
        if isinstance(module, nn.layers.linear.Linear):
            if 'c_proj' in name:
                new_params.append((name + '.weight', residual_init(module.weight)))
            else:
                new_params.append((name + '.weight', normal_init(module.weight)))
            if 'bias' in module:
                new_params.append((name + '.bias', mx.zeros(module.bias.shape)))
        elif isinstance(module, nn.layers.embedding.Embedding):
            new_params.append((name + '.weight', normal_init(module.weight)))
    self = self.update(utils.tree_unflatten(new_params))
```

We add another if statement under our linear branch to check if the nn.Module object has a bias attribute. If it does, we add it to the list initialized to zeros. Finally, we need to transform our list of tuples into a nested dictionary. Luckily mlx has some functions implemented for dealing with parameter dictionaries, and we can use util.tree_unflatten() to convert this list of tuples to a nested parameter dictionary. This is passed into the update method to initialize the parameters. Now we can call _init_parameters() in the constructor.

```
class GPT(nn.Module):
    def __init__(self):
        super().__init__()
        self.wte = nn.Embedding(vocab_size, n_emb) # token embeddings
        self.wpe = nn.Embedding(ctx_len, n_emb) # position embeddings
        self.blocks = nn.Sequential(
            *[Block() for _ in range(n_layers)],
        ) # transformer blocks
        self.ln_f = nn.LayerNorm(dims=n_emb) # final layernorm
        self.lm_head = nn.Linear(n_emb, vocab_size) # output projection
```

```
    self._init_parameters() # <-- initialize params
    # print total number of params on initialization
    total_params = sum([p.size for n,p in utils.tree_flatten(self.parameters())])
    print(f"Total params: {(total_params / 1e6):.3f}M")
# Tensor shapes commented
def __call__(self, x):
    B, T = x.shape # (B = batch_size, T = ctx_len)
    tok_emb = self.wte(x) # (B, T, n_emb)
    pos_emb = self.wpe(mx.arange(T)) # (T, n_emb)
    x = tok_emb + pos_emb # (B, T, n_emb)
    x = self.blocks(x) # (B, T, n_emb)
    x = self.ln_f(x) # (B, T, b_emb)
    logits = self.lm_head(x) # (B, T, vocab_size)
    return logits
def generate(self, max_new_tokens):
    ctx = mx.zeros((1, 1), dtype=mx.int32)
    for _ in range(max_new_tokens):
      logits = self(ctx[:, -ctx_len:]) # pass in last ctx_len characters
      logits = logits[:, -1, :] # get logits for the next token
      next_tok = mx.random.categorical(logits, num_samples=1)
      ctx = mx.concatenate((ctx, next_tok), axis=1)
    return ctx
def _init_parameters(self):
    normal_init = nn.init.normal(mean=0.0, std=0.02)
    residual_init = nn.init.normal(mean=0.0, std=(0.02 / math.sqrt(2 * n_layers)))
    new_params = []
    for name, module in self.named_modules():
        if isinstance(module, nn.layers.linear.Linear):
            if 'c_proj' in name:
                new_params.append((name + '.weight', residual_init(module.weight)))
            else:
                new_params.append((name + '.weight', normal_init(module.weight)))
            if 'bias' in module:
```

```
                new_params.append((name + '.bias', mx.zeros(module.bias.shape)))
            elif isinstance(module, nn.layers.embedding.Embedding):
                new_params.append((name + '.weight', normal_init(module.weight)))
        self = self.update(utils.tree_unflatten(new_params))
```

We also add 2 lines of code in the constructor to print the total number of params. Finally, we are ready to build the training loop.

Training Loop

To train the model we need a loss function. Since we are predicting classes (next token) we use cross-entropy loss.

```
def loss_fn(model, x, y):
    logits = model(x)
    B, T, C = logits.shape # (batch_size, seq_len, vocab_size)
    logits = logits.reshape(B*T, C)
    y = y.reshape(B*T)
    loss = nn.losses.cross_entropy(logits, y, reduction='mean')
    return loss
```

First, we get the logits from the model. Then we reshape logits to make a list of vocab_size length arrays. We also reshape y, the correct token ids, to have the same length. Then we use the built-in cross-entropy loss function to calculate the loss for each example and average them to get a single value.

```
model = GPT()
mx.eval(model.parameters()) # Create the model params (mlx is lazy evaluation)
loss_and_grad = nn.value_and_grad(model, loss_fn)
lr = 0.1
optimizer = optim.AdamW(learning_rate=lr)
```

Next, we instantiate the model, but since mlx is lazy evaluation it wont allocate and create the parameters. We need to call mx.eval on the parameters to ensure they get created. Then we can use nn.value_and_grad() to get a function that returns the loss and gradient of model parameters w.r.t the loss. This is all we need to optimize. Finally, we initialize an AdamW optimizer.

A quick note on nn.value_and_grad(). If you are used to PyTorch you might expect us to use loss.backward() which goes through the computation graph and updates the .grad attribute of each tensor in our model. However, mlx automatic differentiation works on functions instead of computation graphs [3]. Therefore, mlx has built-ins that take in a function and return the gradient function such as nn.value_and_grad() .

Now we define the training loop.

```
num_epochs=20
batch_size=32
for epoch in range(num_epochs):
    model.train(True)
    running_loss = 0
    batch_cnt = 0
    for input, label in get_batches(X_train, y_train, batch_size):
        batch_cnt += 1
        loss, grads = loss_and_grad(model, input, label)
        optimizer.update(model, grads)
        running_loss += loss.item()
        # compute new parameters and optimizer state
        mx.eval(model.parameters(), optimizer.state)
    avg_train_loss = running_loss / batch_cnt
    model.train(False) # set eval mode
    running_loss = 0
    batch_cnt = 0
    for input, label in get_batches(X_val, y_val, batch_size):
        batch_cnt += 1
        loss = loss_fn(model, input, label)
        running_loss += loss.item()
    avg_val_loss = running_loss / batch_cnt
    print(f"Epoch {epoch:2} | train = {avg_train_loss:.4f} | val = {avg_val_loss:.4f}")
```

The outer loop runs through the epochs. We first set the model to training mode because some

modules have different behaviors during training and testing such as dropout. Then we use our get_batches function from earlier to loop through batches of the training data. We get the loss over the batch and the gradient using loss_and_grad . Then we pass the model and gradients to the optimizer to update the model parameters. Finally we call mx.eval (remember mlx does lazy evaluation) to ensure the parameters and optimizer state get updated. Then we calculate the average train loss over the data to print later. This is one pass through the training data. Similarly, we calculate the validation loss and then print the average train and val loss over the epoch.

```
completion = decode(model.generate(1000)[0].tolist())
print(completion)
with open('completions.txt', 'w') as f:
    f.write(completion)
```

Finally, we add some code to generate from our model. Since the generation output is still in the (B, T) shape we have to index it at 0 to make it 1D and then convert it from an mlx array to a Python list. Then we can pass it to our decode function from earlier, and write it to a file.

These are the parameters we will use for training (you can play around with this):

```
ctx_len = 128
n_emb = 128
dropout = 0.1
head_size = 128
n_heads = 4
n_layers = 3
num_epochs = 20
batch_size = 64
lr = 1e-3
```

Now we can run the file to start training. With the settings above training took around 10 minutes on my m2 MacBook. I achieved the following training loss last epoch.

```
Epoch 19 | train = 1.6961 | val = 1.8143
```

Lets look at some output.

GLOUCESTER:
But accomes mo move it.

KING EDWARD:
Where our that proclaim that I curse, or I sprithe.

CORIOLANUS:
Not want:
His bops to thy father
At with hath folk; by son and fproathead:
The good nor may prosperson like it not,
What, the beggares
More hath, when that made a,
Your vainst Citizen:
Let here are go in queen me and knife
To my deserved me you promise: not a fettimes,
That one the will not.

CORIOLANUS:
And been of queens,
Thou to do we best!

JULIET:
Not, brother recourable this doth our accuse
Into fight!

Not bad for just 10 minutes of training with a tiny model that is predicting characters! It clearly has the form of Shakespeare, although it is nonsense. The only difference between our model and the real GPT-2 now is scale! Now I encourage you to experiment try out different settings, maybe tinker with the architecture, and see how low of a loss you can achieve.

References

[1] Karpathy A (2015).Tiny Shakespeare [Data set]. https://github.com/karpathy/char-rnn (MIT license)

[2] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, Language Models are Unsupervised Multitask Learners (2019), OpenAI

[3] Automatic Differentiation mlx docs