



DeepLearning.AI

Introduction to Generative AI

What is generative AI?

Please put Luna in a scene where she is running through a forest early in the morning



Here is Luna running through a forest early in the morning:



If you would like any further modifications or additional scenes with Luna, please let me know! You can refer to the details of this image using this reference:

Reference 2

please put Luna in a scene where she's playing with two cute foxes in an enchanted forest early in the morning



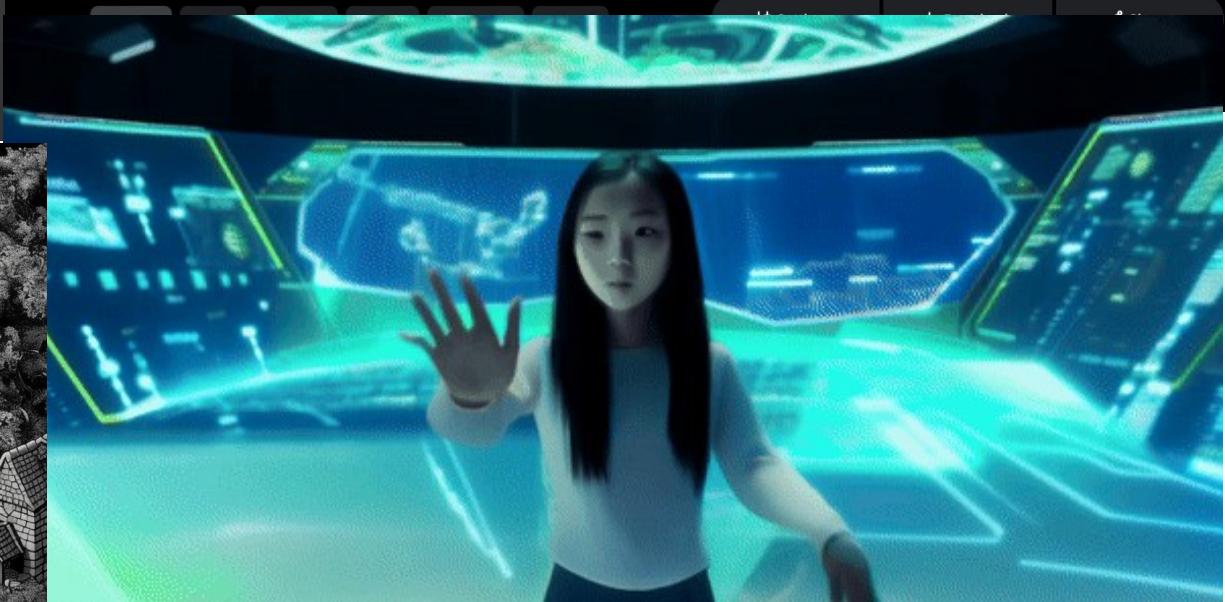
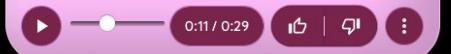
MusicFX



irish jig ▾ with melancholy ▾ undertones



IRISH JIG WITH MELANCHOLY UNDERTONES



```
1 //dictionary of 2 letter ISO country codes and their corresponding country
```



Please draw an image of a time travelling caticorn -- a creature that's a cross between a cat and a unicorn, with a cat's face, but a Unicorn's winged horse-like body. Give it a mane of rainbow colored-hair, and a shimmering horn. Make it CGI-like to look like it's from a Dreamworks movie.



Here's the CGI-like time travelling caticorn with a cat's face, a unicorn's winged horse-like body, a mane of rainbow-colored hair, and a shimmering horn. It looks like it could be from a Dreamworks movie! If there's anything you'd like to adjust or add, let me know!

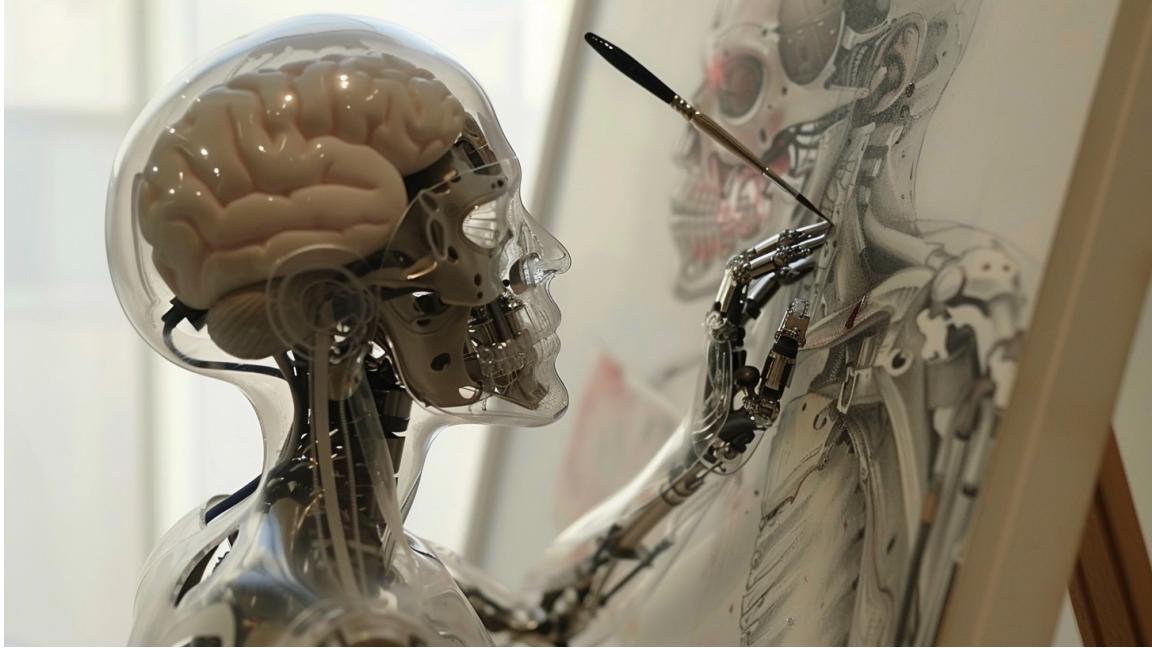


DeepLearning.AI

Introduction to Generative AI

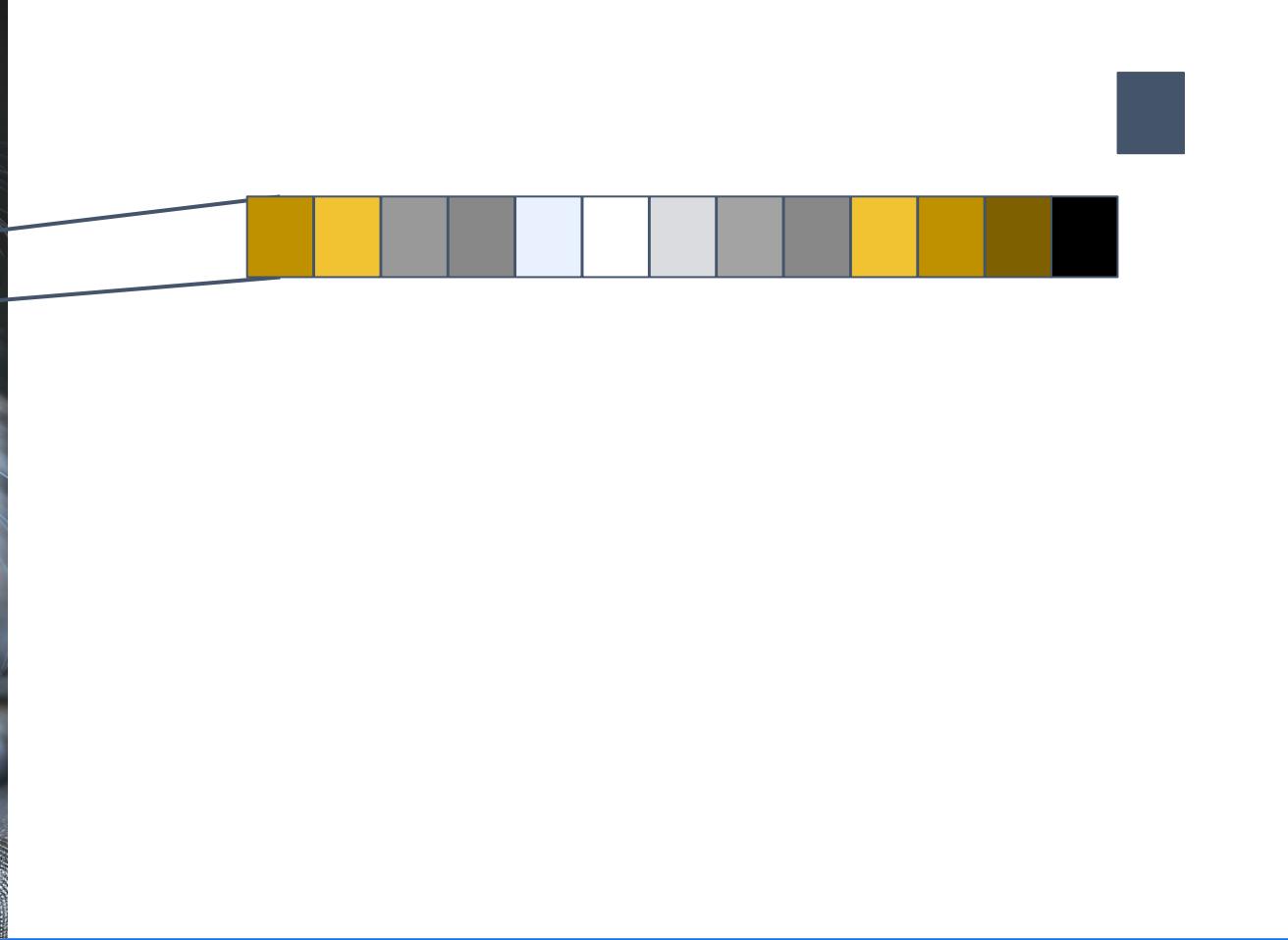
AI and machine learning

What is Artificial Intelligence?



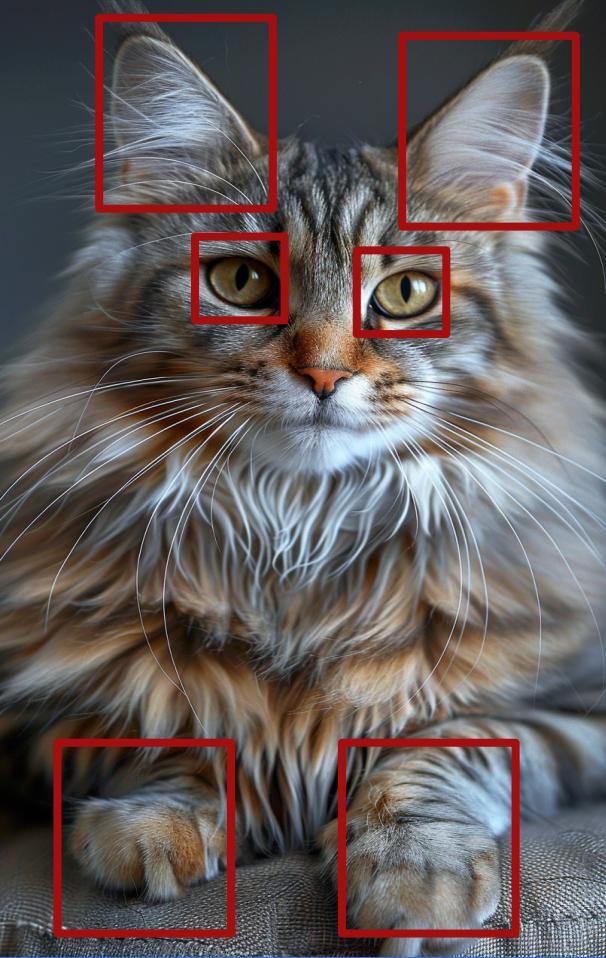
The concept of programming a computer to react to data in a similar way to a thinking being.





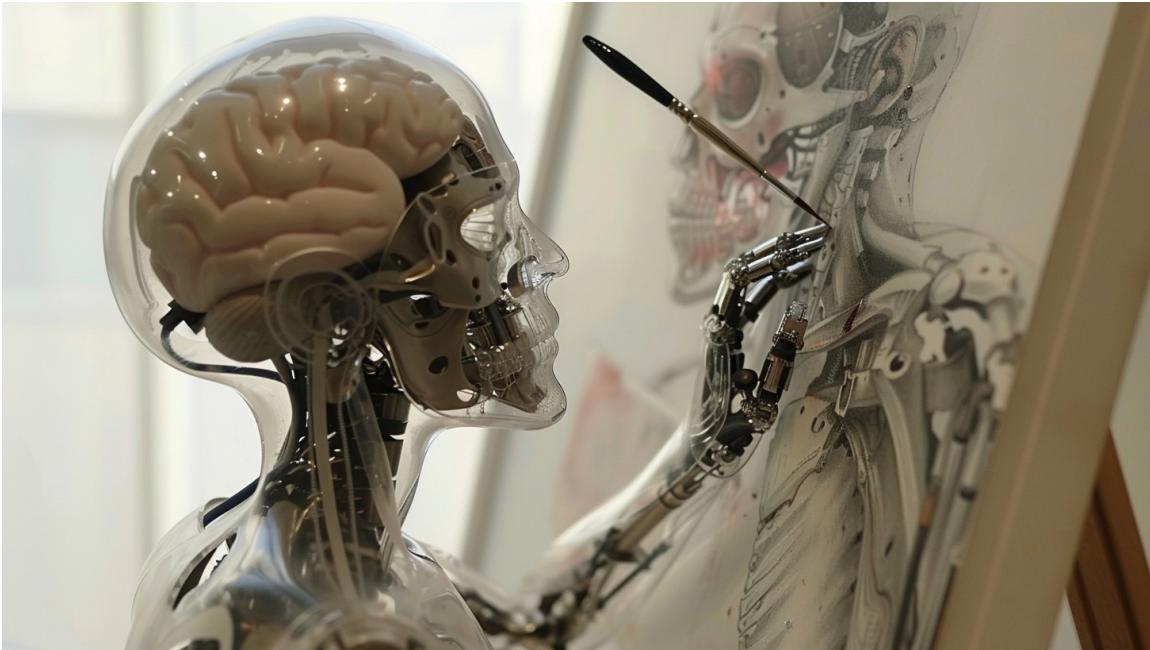


“Cat”



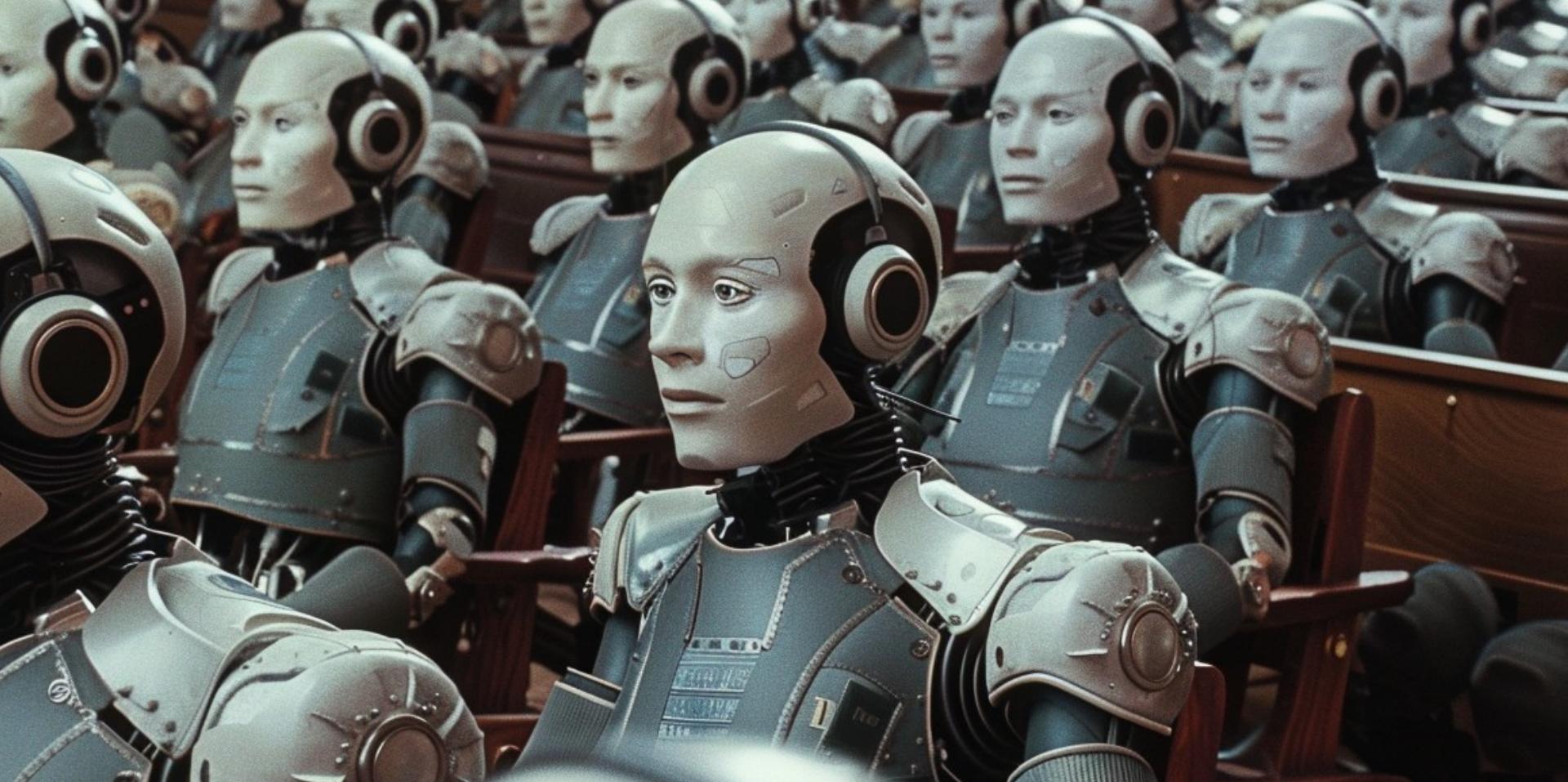
“Features”

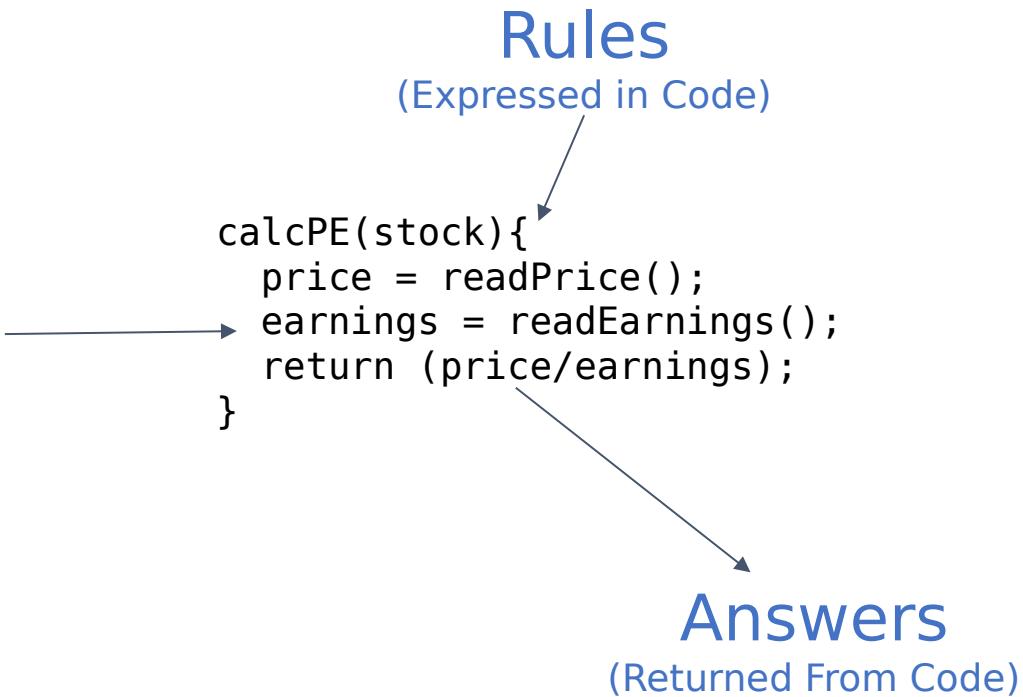
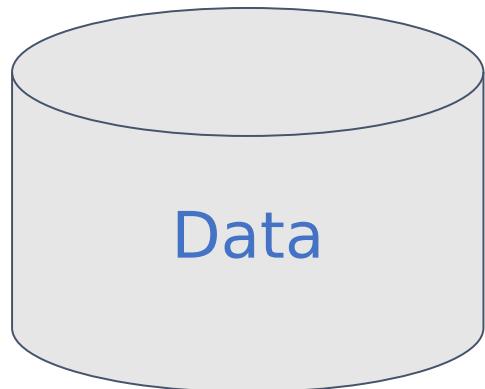
What is Artificial Intelligence?



The concept of programming a computer to react to data in a similar way to a thinking being.

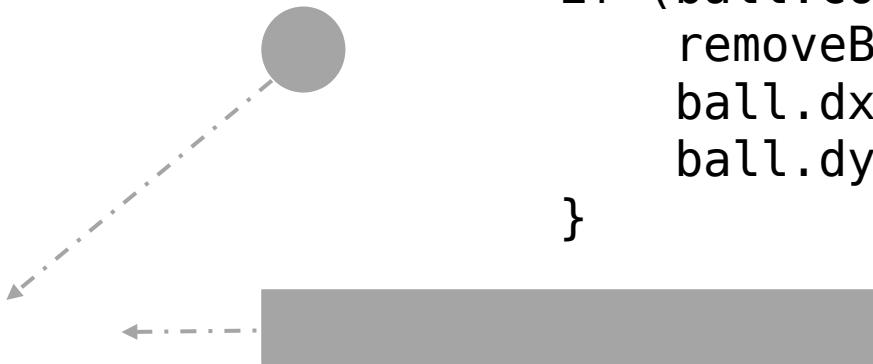


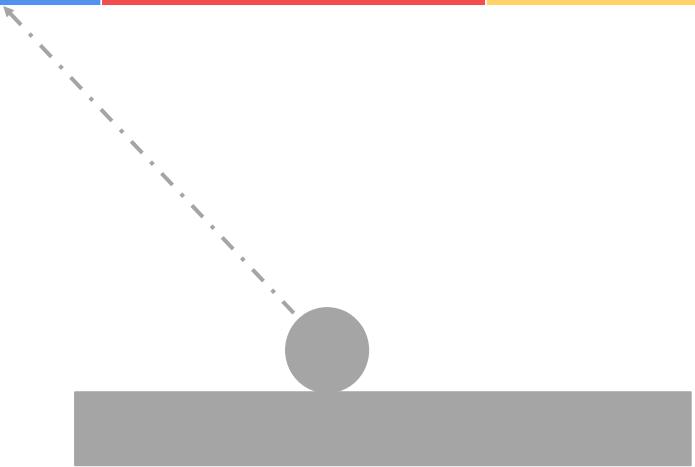




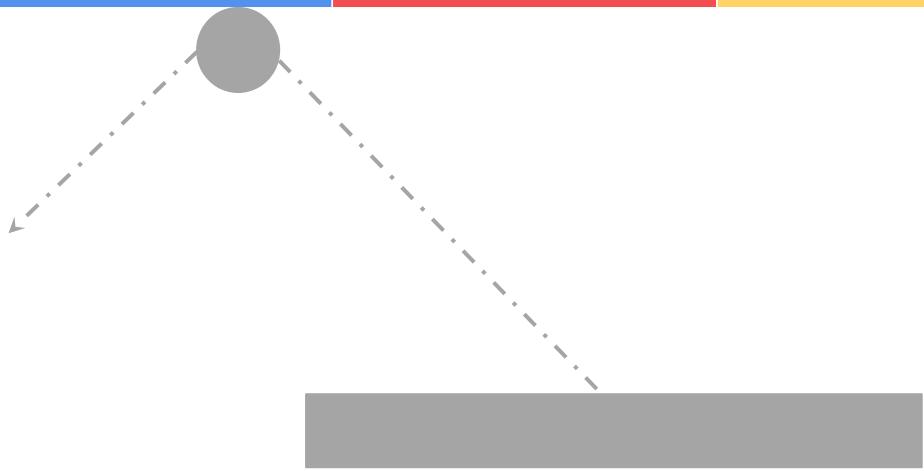


```
if (ball.collide(brick)){
    removeBrick();
    ball.dx=-1*(ball.dx);
    ball.dy=-1*(ball.dy);
}
```

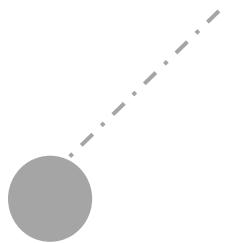




```
if (ball.collide(brick)){  
    removeBrick();  
    ball.dx=-1*(ball.dx);  
    ball.dy=-1*(ball.dy);  
}
```



```
if (ball.collide(brick)){  
    removeBrick();  
    ball.dx=-1*(ball.dx);  
    ball.dy=-1*(ball.dy);  
}
```



```
if (ball.collide(brick)){
    removeBrick();
    ball.dx=-1*(ball.dx);
    ball.dy=-1*(ball.dy);
}
```







DeepLearning.AI

Introduction to Generative AI

Machine learning example

Activity recognition



```
if(speed<4){  
    status=WALKING;  
}
```



```
if(speed<4){  
    status=WALKING;  
} else {  
    status=RUNNING;  
}
```



```
if(speed<4){  
    status=WALKING;  
} else if(speed<12){  
    status=RUNNING;  
} else {  
    status=BIKING;  
}
```



// ????



Activity recognition



01010010101001010
10100101010100101
11010100101010010
10100101010010101
00101010
Label =
WALKING



10101001010010101
01010101001001001
00010010011111010
10111110101001001
11101011111101111
Label =
RUNNING



10010100111110101
01110101011101010
11101010101111010
10101111111100011
11010101
Label = BIKING



1111111110100111
01001111101011111
01010101110101010
10111010101010101
00111110
Label =
GOLFING

Activity recognition



01010010101001010
10100101010100101
11010100101010010
10100101010010101
00101010
Label =
WALKING



10101001010010101
01010101001001001
00010010011111010
10111110101001001
11101011
Label =
RUNNING



10010100111110101
01110101011101010
11101010101111010
10101111111100011
11010101
Label =
BIKING

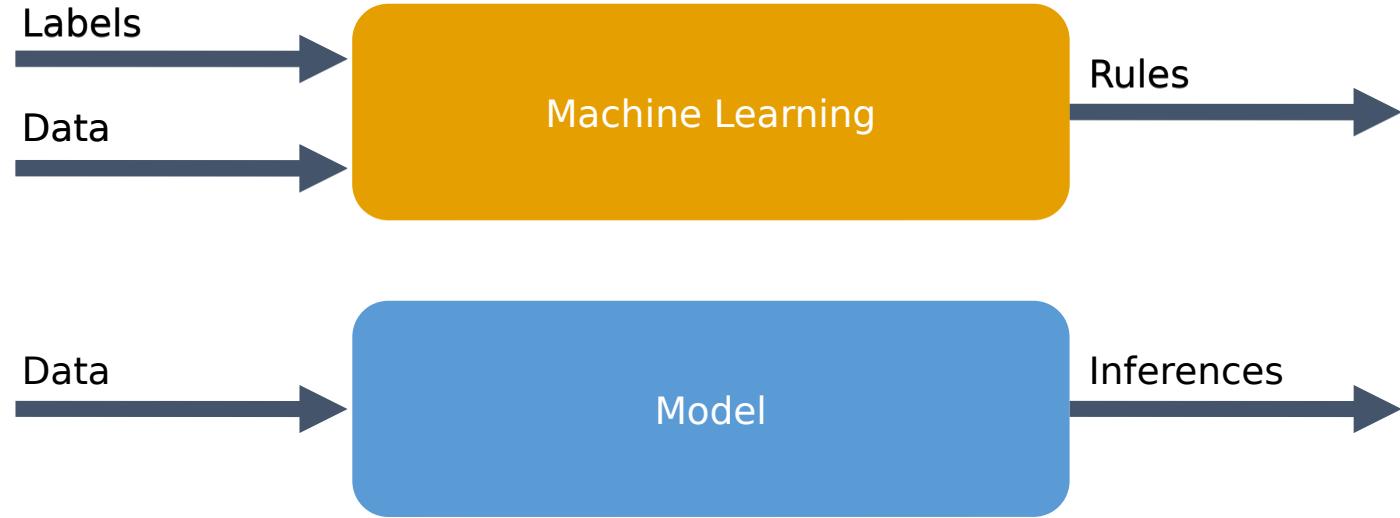


1111111110100111
01001111101011111
01010101110101010
10111010101010101
00111110
Label =
GOLFING

The machine learning paradigm



The machine learning paradigm



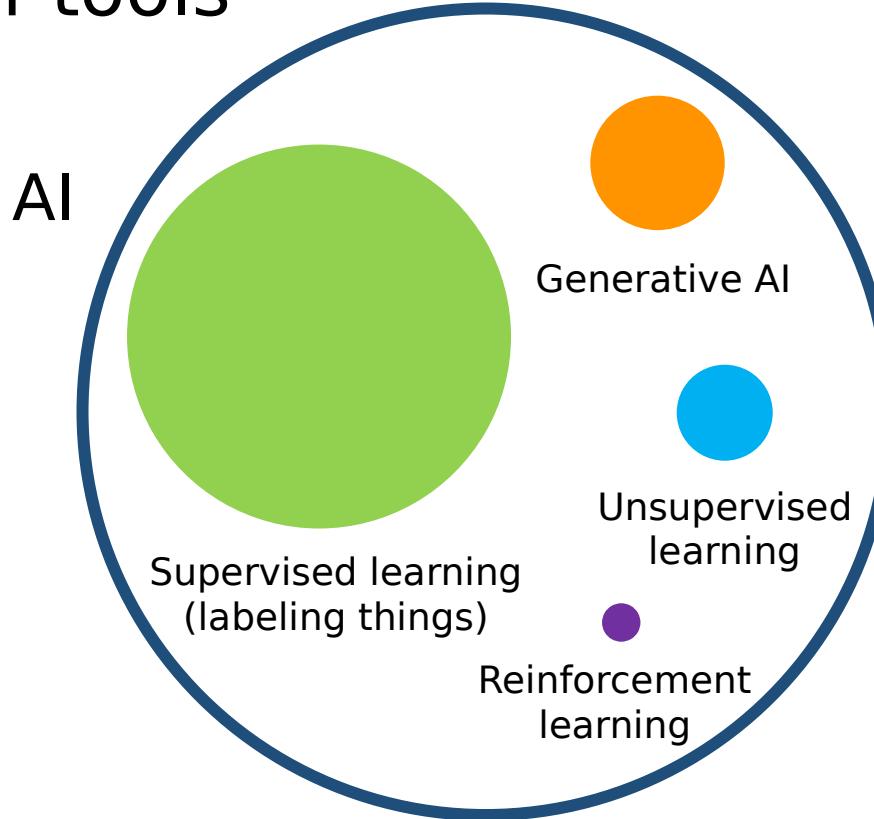


DeepLearning.AI

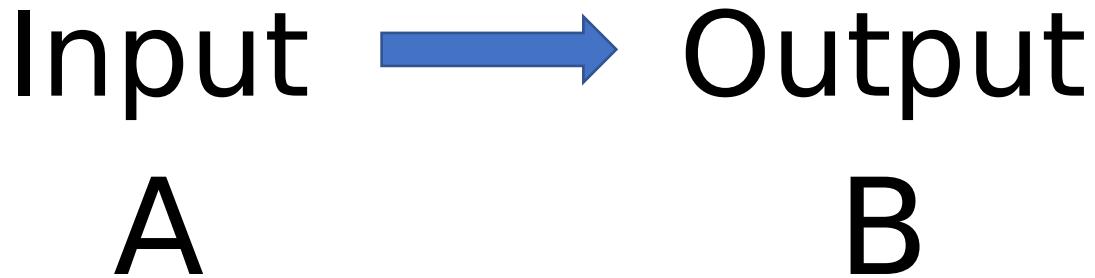
Introduction to Generative AI

Supervised learning

AI is a set of tools



Supervised learning (labeling things)

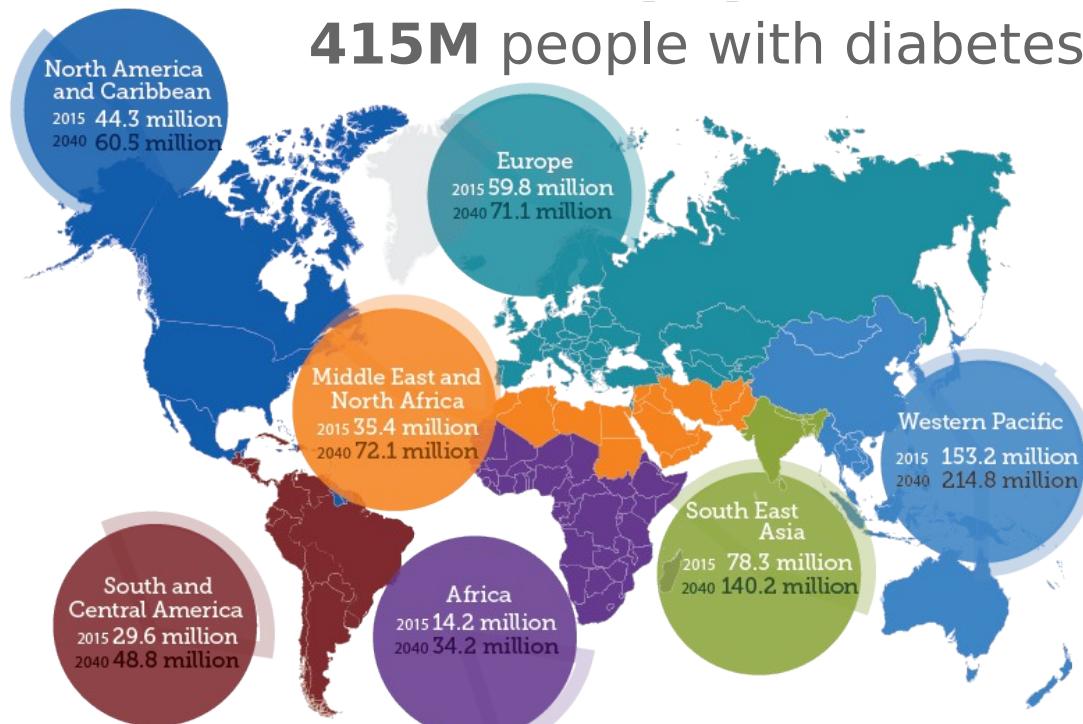


Supervised learning (labeling things)

Input (A)	Output (B)	Application
Email	Spam? (0/1)	Spam filtering
Ad, user info	Click? (0/1)	Online advertising
Image, radar info	Position of other cars	Self-driving car
Medical image	Diagnosis	Healthcare
Image of phone	Defect? (0/1)	Visual inspection
Audio recording	Text transcript	Speech recognition
Restaurant reviews	Sentiment (pos/neg)	Reputation monitoring

Diabetic retinopathy: fastest growing cause of blindness

415M people with diabetes



Source: International Diabetes Federation 2015 Atlas, www.idf.org/e-library/epidemiology-research/diabetes-atlas/

Diabetic retinopathy: fastest growing cause of blindness

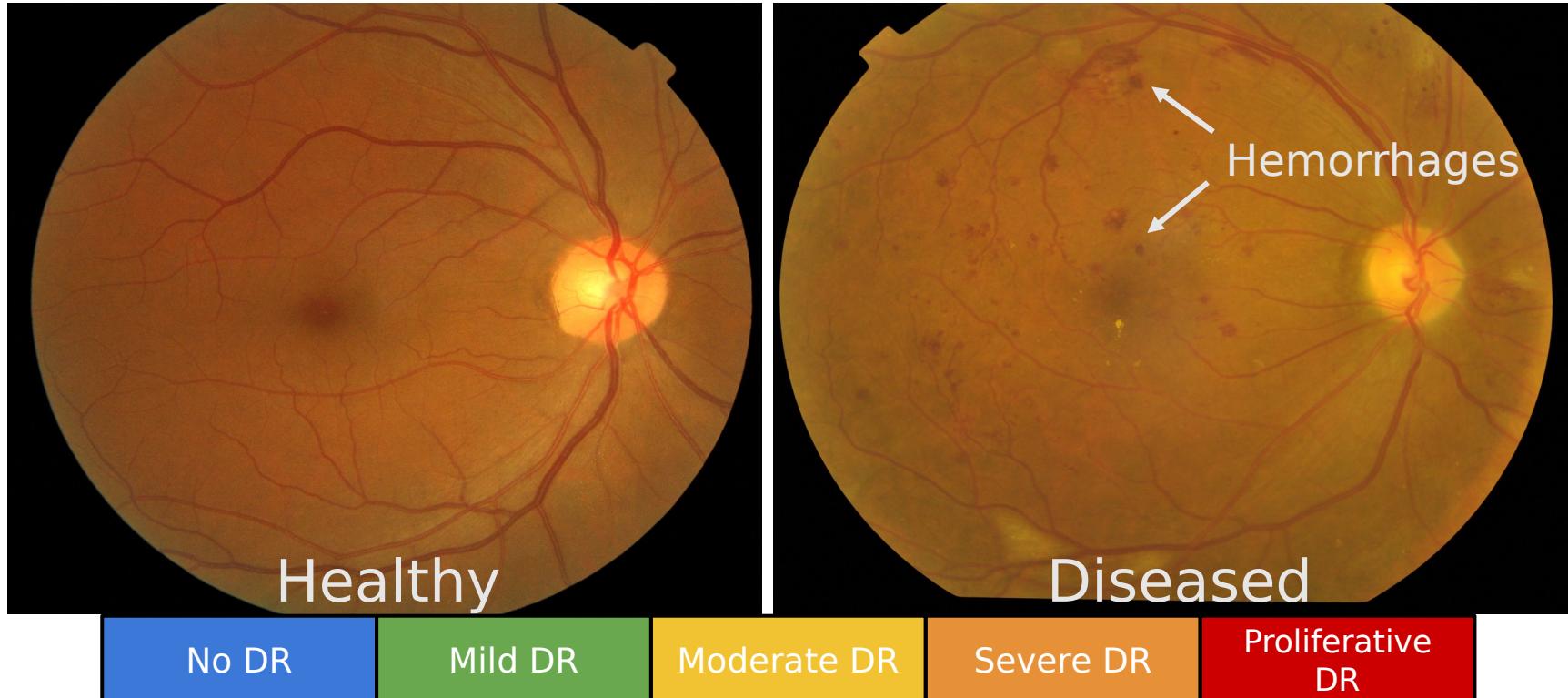
Regular screening is key to preventing blindness



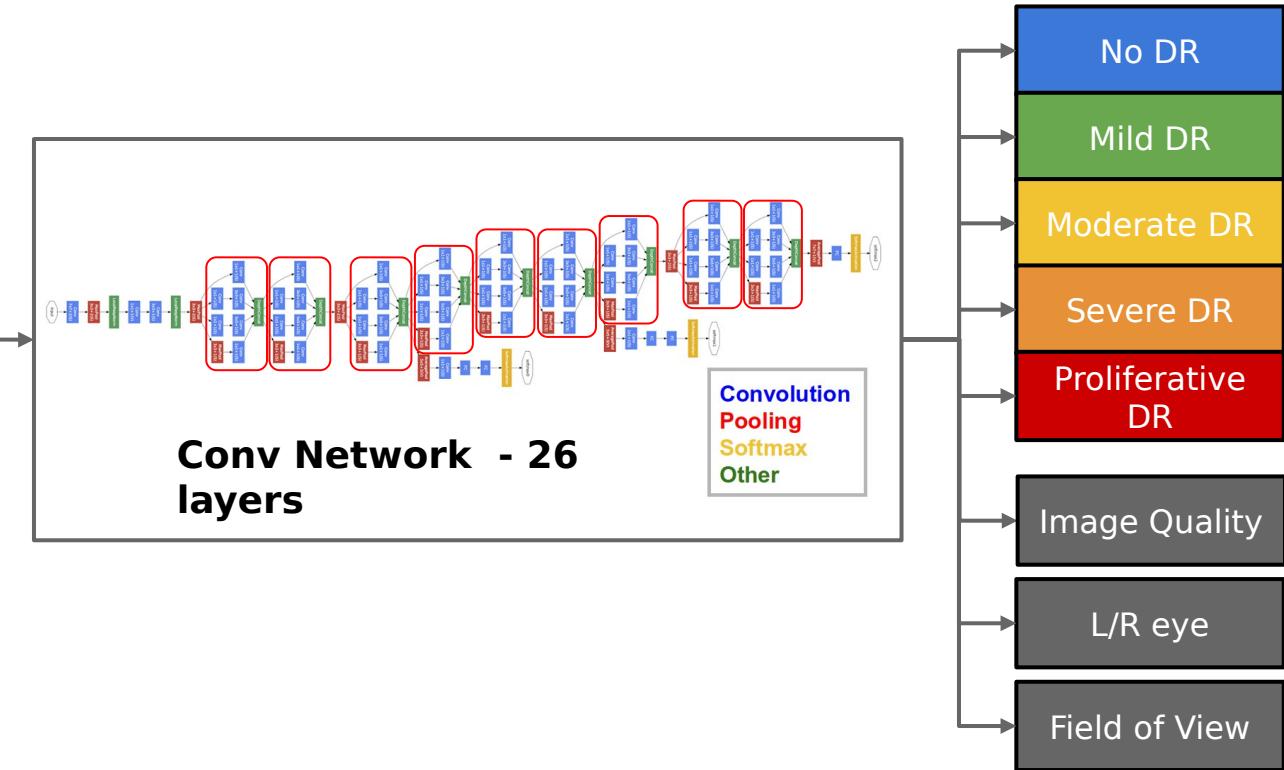


INDIA
Shortage of 127,000 eye doctors
45% of patients suffer vision loss before diagnosis.

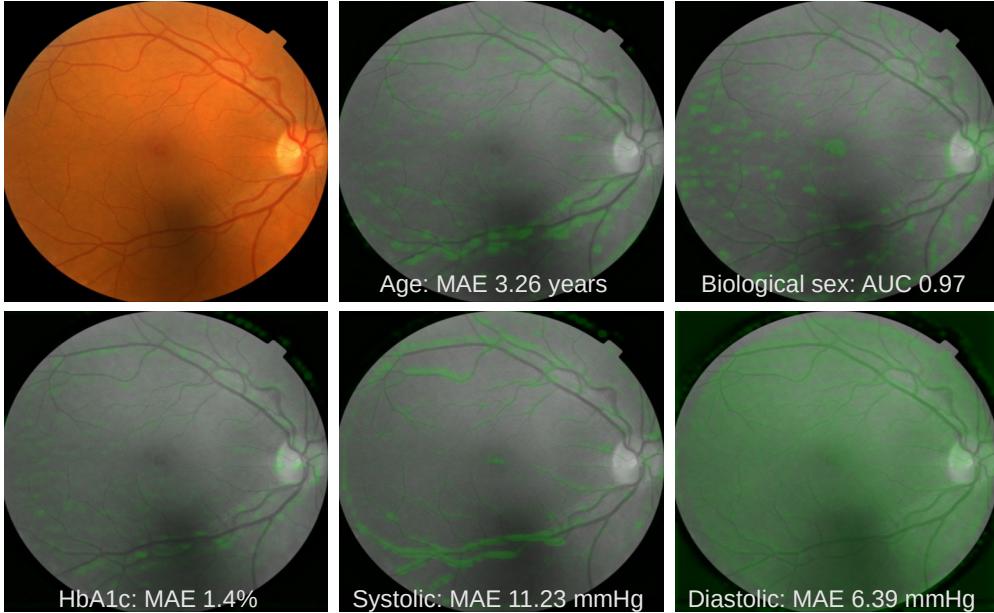
How DR is diagnosed: Retinal fundus images



Adapt deep neural network to read fundus images



Completely new, novel scientific discoveries

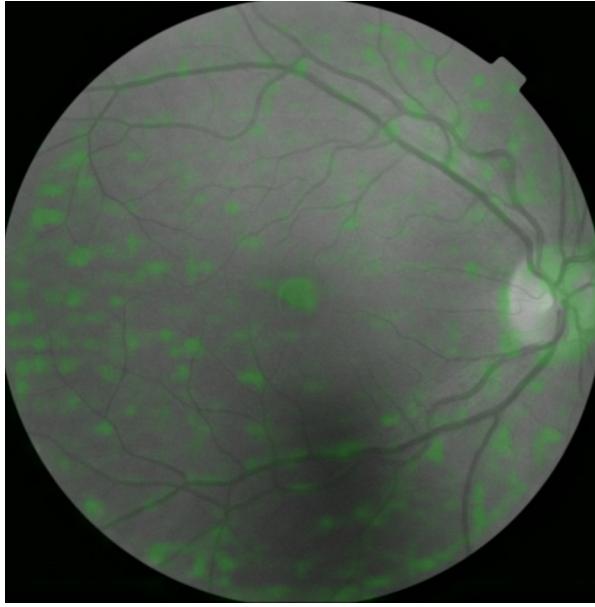


Machine learning can sometimes find other relationships that doctors can't see. For example

- **Patterns in retinal images that depend on age, gender, and other factors**
- **Trends in retinal images features and cardiovascular risk**

R. Poplin, A. Varadarajan et al. "Predicting Cardiovascular Risk Factors from Retinal Fundus Photographs using Deep Learning." *Nature Biomedical Engineering*, 2018.

Completely new, novel scientific discoveries



Self-reported sex?

Ophthalmologists can't do this, so should be no better than flipping a coin (i.e. AUC of 0.50)

How Large Language Models (LLMs) work

LLMs are built by using supervised learning ($A \rightarrow B$) to repeatedly predict the next word.

My favorite dessert is chocolate chip ice cream

Input (A)	Output (B)
My favorite dessert is	chocolate
My favorite dessert is chocolate	chip
My favorite dessert is chocolate chip	ice
My favorite dessert is chocolate chip ice	cream

When we train a very large AI system on a lot of data (hundreds of billions of words), we get a Large Language Model like ChatGPT.



Introduction to Generative AI

Introduction to transformers

Attention Is All You Need

Ashish Vaswani*
Google Brain
avaswani@google.com

Noam Shazeer*
Google Brain
noam@google.com

Niki Parmar*
Google Research
nikip@google.com

Jakob Uszkoreit*
Google Research
usz@google.com

Llion Jones*
Google Research
llion@google.com

Aidan N. Gomez* †
University of Toronto
aidan@cs.toronto.edu

Lukasz Kaiser*
Google Brain
lukaszkaiser@google.com

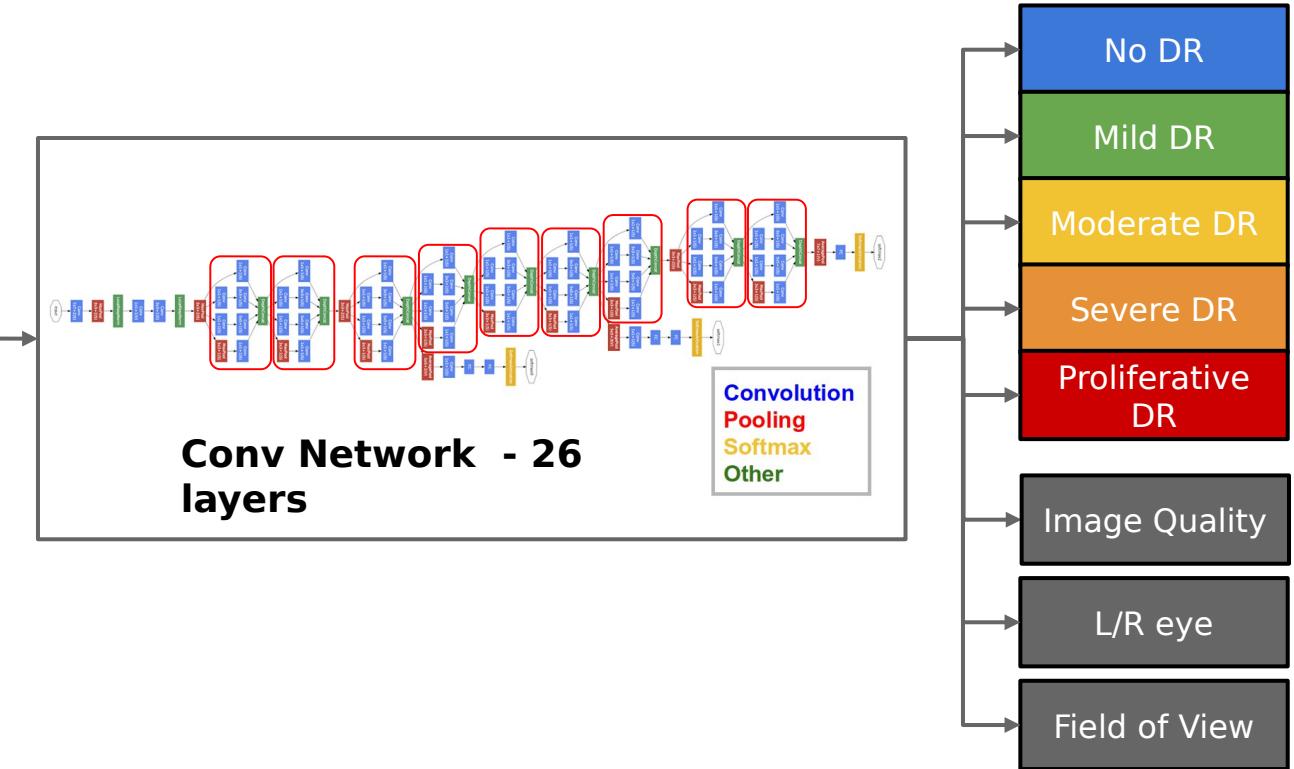
Illia Polosukhin* †
illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

Vaswani et al. 2017
arXiv:1706.03762

Supervised learning



In Ireland, I went to secondary school,
so I had to study _____

In **Ireland**, I went to **secondary school**,
so I had to study Gaelic.



DeepLearning.AI

Introduction to Generative AI

Key transformer
concepts

Key transformer concepts

1. Attention

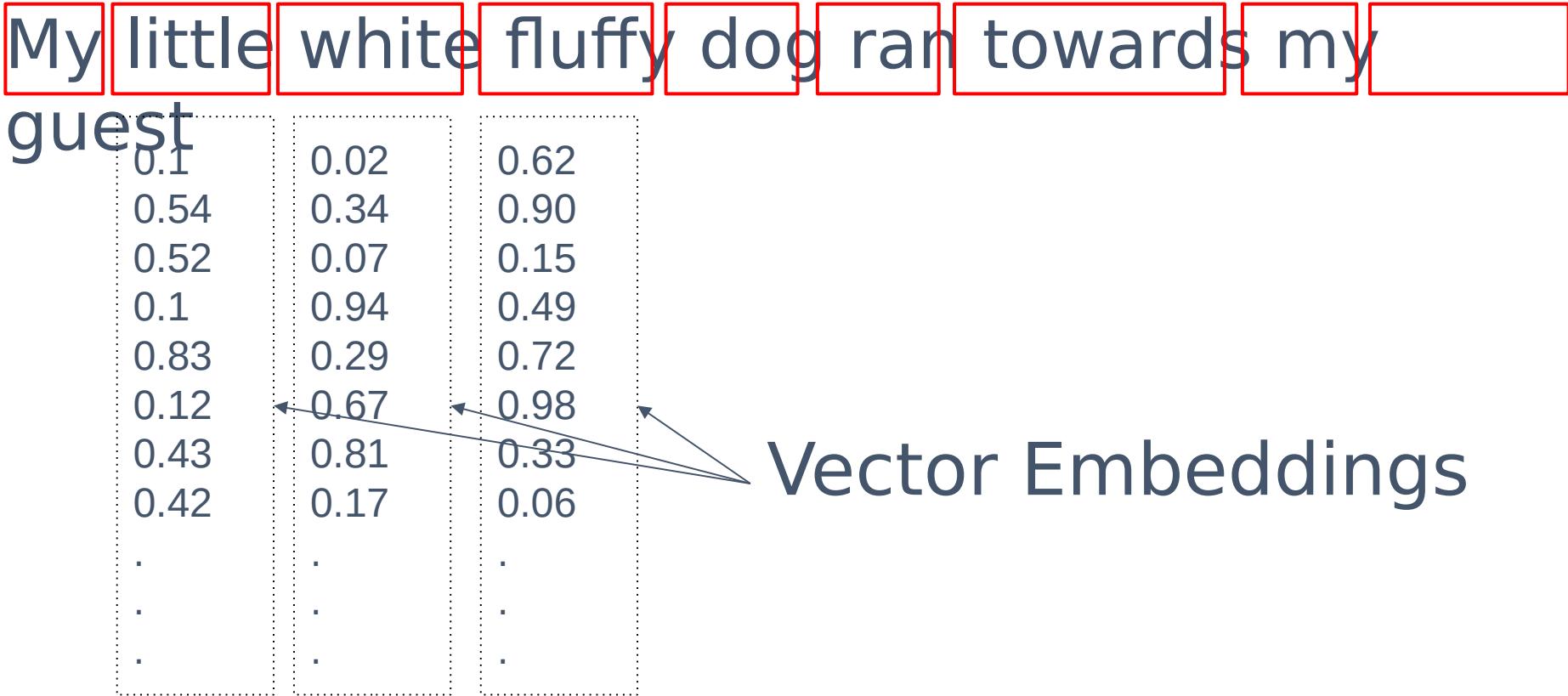
Allows the model to focus on specific words when predicting the next word.

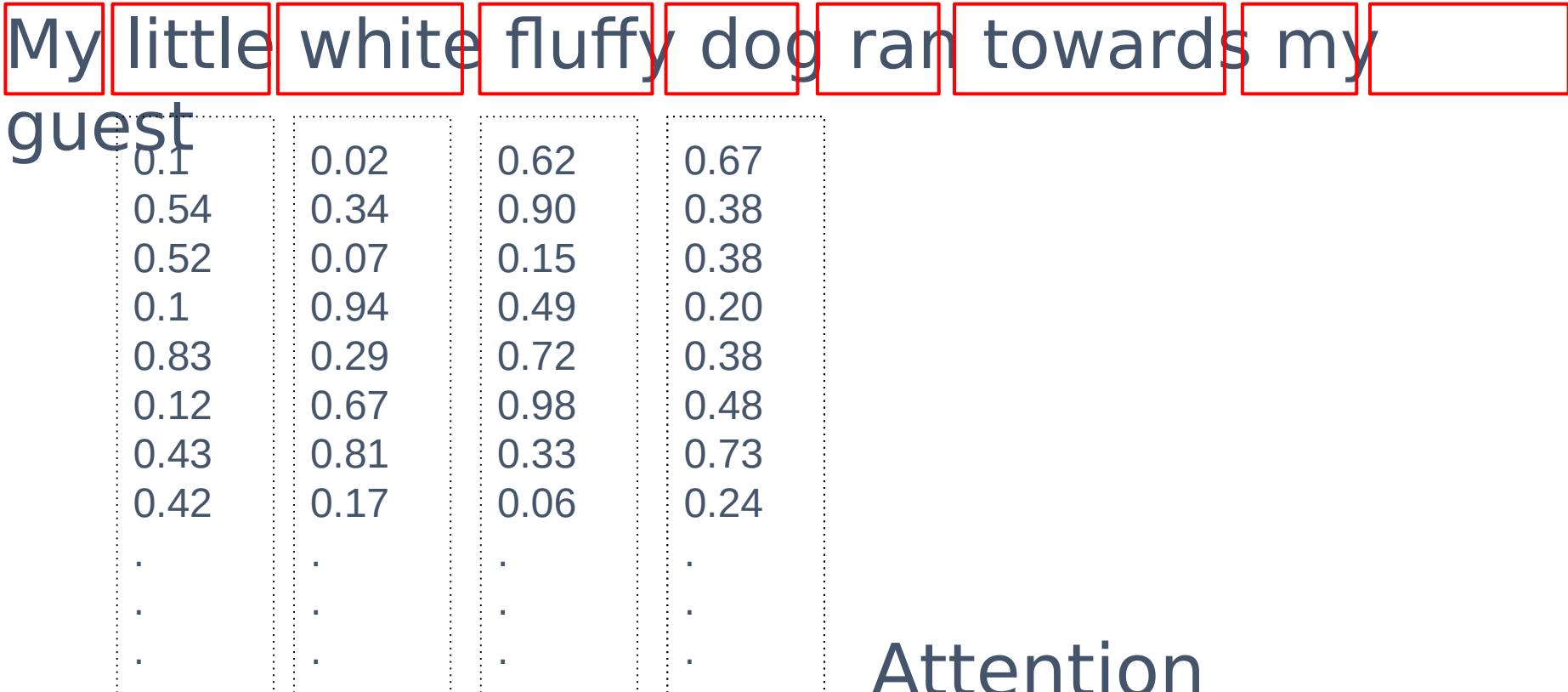
My **little** **white** **fluffy** dog ran towards my guest

My little white fluffy dog ran towards my

guest

Tokens





Key transformer concepts

1. Attention

Allows the model to focus on specific words when predicting the next word.

2. Encoders and Decoders

Components of the model architecture that prepare and process text.

```
def do_something_with(param):  
    return param**2
```

```

import re
import warnings

from keras.src import backend
from keras.src import initializers
from keras.src import optimizers
from keras.src import schedules import learning_rate_schedule
from keras.src.saving import serialization_lib
from keras.src.saving.legacy import KerasSaveable
from keras.src.utils import tracking
from keras.src.utils import auto_name

class BaseOptimizer(KerasSaveable):
    def __init__(self,
                 learning_rate,
                 weight_decay=None,
                 clipnorm=None,
                 clipvalue=None,
                 global_clipnorm=None,
                 use_ema=False,
                 ema_momentum=0.99,
                 ema_overwrite_frequency=None,
                 loss_scale_factor=None,
                 gradient_accumulation_steps=None,
                 name=None,
                 **kwargs,
                 ):
        self._lock = False

    if kwargs.pop("decay", None) is not None:
        warnings.warn(
            "'named' 'decay' is no longer supported and will be ignored."
        )
    if kwargs:
        raise ValueError(f"Arguments not recognized: {list(kwargs)}")

    if name is None:
        name = auto_name(self, class_=__name__)
    self.name = name
    self.weight_decay = weight_decay
    self.clipnorm = clipnorm
    self.global_clipnorm = global_clipnorm
    self.clipvalue = clipvalue
    self.use_ema = use_ema
    self.loss_scale_factor = loss_scale_factor
    self.gradient_accumulation_steps = gradient_accumulation_steps

    if gradient_accumulation_steps:
        if not gradient_accumulation_steps == 2:
            raise ValueError(f"gradient_accumulation_steps must be an integer >= 2. "
                            f"(Received: gradient_accumulation_steps={gradient_accumulation_steps})")
    else:
        dtype=reference_variable.dtype,
        name=name,
        )

def check_variables_are_known(self, variables):
    for v in variables:
        if self._var_key(v) not in self._trainable_variables_indices:
            raise ValueError(f"Variable {v}: This optimizer can only "
                            "be called for the variables it was originally built with. "
                            "When working with a new set of variables, you should "
                            "recreate a new optimizer instance.")

def assign(self, variable, value):
    """Assign a value to a variable.

    This should be used in optimizers instead of `variable.assign(value)` to
    support backend specific optimizations.
    Note that the variable can be a model variable or an optimizer variable;
    it can be a backend native variable or a Keras variable.

    Args:
        variable: The variable to update.
        value: The value to add to the variable.
        variable.assign(value)
    """
    # Add a value to a variable.

    This should be used in optimizers instead of
    'variable.assign_add(value)' to support backend specific optimizations.
    Note that the variable can be a model variable or an optimizer variable;
    it can be a backend native variable or a Keras variable.

    Args:
        variable: The variable to update.
        value: The value to add to the variable.
        variable.assign_add(value)
    """
    # Add a value to a variable.

    def assign_add(self, variable, value):
        """Add a value to a variable.

        This should be used in optimizers instead of
        'variable.assign_sub(value)' to support backend specific optimizations.
        Note that the variable can be a model variable or an optimizer variable;
        it can be a backend native variable or a Keras variable.

        Args:
            variable: The variable to update.
            value: The value to subtract from a variable.
            variable.assign_sub(value)
        """
        # Subtract a value from a variable.

        This should be used in optimizers instead of
        'variable.assign_sub(value)' to support backend specific optimizations.
        Note that the variable can be a model variable or an optimizer variable;
        it can be a backend native variable or a Keras variable.

        Args:
            variable: The variable to update.
            value: The value to subtract from a variable.
            variable.assign_sub(value)
        """

    self._track_variable(iterations)
    self._iterations = iterations

    # Create learning rate (schedule or variable)
    if isinstance(learning_rate, learning_rate_schedule.LearningRateSchedule):
        self._learning_rate = learning_rate.schedule
    elif callable(learning_rate):
        self._learning_rate = learning_rate()
    else:
        self._learning_rate = learning_rate
    self._learning_rate = learning_rate
    self._learning_rate = learning_rate
    else:
        if not isinstance(learning_rate, float):
            raise ValueError(
                f"Expected learning_rate to be float, or an instance of "
                f"LearningRateSchedule, or a callable "
                f"(that takes in the current iteration value). "
                f"(and returns the corresponding learning_rate value). "
                f"(Received instead: learning_rate={learning_rate})"
            )
        with backend.name_scope(self.name, caller=self):
            learning_rate = backend.Variable(
                learning_rate,
                name="learning_rate",
                dtype=backend.floatx(),
                trainable=True,
                aggregation="only_first_replica",
            )
            self._track_variable(learning_rate)
            self._learning_rate = learning_rate
    self._track_variable(variable)
    self._tracker.add("variables", variable)

    # Tracking no automatic dependency tracking
    def build(self, variables):
        if self.use_ema:
            for v in variables:
                moving_average = []
                if self.gradient_accumulation_steps:
                    self._accumulated_gradients = []
                    for i, v in enumerate(variables):
                        trainable_variables_indices[v.key.variable] = i
                    self._mean_variables_moving_average.append(
                        self._add_variable_from_reference(
                            variable,
                            name="average",
                        )
                    )
                    self._gradient_accumulation_steps = self._accumulated_gradients.append(
                        self._add_variable_from_reference(
                            variable,
                            name="gradient_accumulator",
                        )
                    )
                else:
                    self._trainable_variables = variables[]
                    self._built = True
        def var_key(self, variable):
            """Get a stable ID for the variable and the variable instance mapping.
            return id(variable)
            """
            @property
            def variables(self):
                return self._variables

            def get_variable(self, variable):
                return self._trainable_variables_indices[self._var_key(variable)]

            def add_variable(
                self,
                shape,
                initializer="zeros",
                dtype=None,
                aggregation="mean",
                name=None,
            ):
                self._check_super_called()
                initializers = initializers.get(initializer)
                with backend.name_scope(self.name, caller=self):
                    variable = backend.Variable(
                        initializer,
                        shape,
                        dtype=dtype,
                        trainable=False,
                        aggregation=aggregation,
                        name=name,
                    )
                    self._track_variable(variable)
                    return variable

            def add_variable_from_reference(
                self,
                reference_variable,
                name=None,
                initializer="zeros",
            ):
                """Add an all-zeros variable with the shape and dtype of a reference
                variable.
                name = name + v.name
                if not reference_variable.path:
                    name = reference_variable.path + name
                else:
                    name = (
                        str(reference_variable.name).replace("/", ".") + "." + name
                    )
                + name
                return self._add_variable(
                    shape=reference_variable.shape,
                    dtype=reference_variable.dtype,
                    aggregation=reference_variable.aggregation,
                    name=name,
                )

            def update_step(self, gradient, variable, learning_rate):
                if not isinstance(gradient, Variable):
                    raise NotIterableError(
                        f"Expected gradient to be Variable, "
                        f"but got {type(gradient)}."
                    )
                grads, trainable_variables = zip(*grads_and_vars)
                self._apply_grads(trainable_variables)
                return self._iterations

            def apply_gradients(self, grads, grads_and_vars):
                grads, trainable_variables = zip(*grads_and_vars)
                self._apply_grads(trainable_variables)
                grads should be a list of gradient tensors
                with 1:1 mapping to the list of variables the optimizer was built with.
                trainable_variables can be provided
                on the first call to build the optimizer.
                ...
                if len(grads) == 0:
                    # It is possible that the grad is empty. In this case,
                    # apply_gradients is a no-op.
                    return
                if len(grads) != len(self._trainable_variables):
                    raise ValueError(
                        f"Expected gradients to have length {len(self._trainable_variables)}, "
                        f"but got {len(grads)}. "
                        f"Optimizer was built with {len(self._trainable_variables)} variables. "
                        f"Please make sure your gradients list has the right length. "
                        f"(Received: len(grads)={len(grads)}) gradients, but the "
                        f"Optimizer is tracking {len(self._trainable_variables)} "
                        f"trainable variables."
                    )
                trainable_variables = self._trainable_variables
                grads = list(trainable_variables)
                # Optionally build optimizer.
                if not self._built:
                    with backend.name_scope(self.name, caller=self):
                        self._build(trainable_variables)
                    self._built = True
                self._check_variables_are_known(trainable_variables)

                with backend.name_scope(self.name, caller=self):
                    # Only build optimizer if there are gradients with their gradients if
                    # their overwrite with gradient is set.
                    grads, trainable_variables = (
                        self._overwrite_variables_directly_with_gradients(
                            grads, trainable_variables
                        )
                    )
                    # Filter empty gradients.
                    grads, trainable_variables = self._filter_empty_gradients(
                        grads, trainable_variables
                    )
                    if len(list(grads)) == 0:
                        return
                    # Unscale gradients.
                    scale = self._scale_factor
                    if scale is None:
                        grads = [g if g is None else g / scale for g in grads]
                    # Apply clipping and weight decay
                    grads = self._clip_gradients(grads)
                    self._apply_weight_decay(trainable_variables)

                    # Apply gradient updates.
                    self._backend_apply_gradients(grads, trainable_variables)
                    # Apply gradient accumulation by applying gradients.
                    for variable in trainable_variables:
                        if variable.constraint is not None:
                            variable.assign_variable.constraint(variable)

                    self._backend_apply_gradients(self, grads, trainable_variables)
                    # apply method that can be overridden by different backends.
                    JAX overrides it in order to deal with statelessness in gradient
                    accumulation and EMA handling.

                    The below implementation is intended to be generally backend-agnostic,
                    but may not work with all backends.

                    This method does 4 things:
                    - Call the optimizer's update_step() to update trainable variables
                    - Update trainable variables
                    - Update EMA variables, if EMA is configured
                    - Update gradient accumulators, if gradient accumulation is configured
                    ...
                    if self.gradient_accumulation_steps:
                        is_update_step = 1
                        if self._iterations % 1 == 0:
                            # % self.gradient_accumulation_steps == 0
                            # trainable_variables might have been filtered in previous
                            # processing steps, so we need to do a correct mapping between
                            # self._accumulated_gradients and trainable_variables
                            acc_grads = []
                            self._accumulated_gradients(self._get_variable_index(v))
                            for v in trainable_variables
                            acc_grads += [
                                self._accumulated_gradients(self._get_variable_index(v))
                                for v in trainable_variables
                            ]
                            self._update_step(acc_grads, trainable_variables)
                            # Run update step with accumulated grads + reset accumulators
                            steps = self.gradient_accumulation_steps
                            ...
                        else:
                            self._update_step(grads, trainable_variables)
                            # Run update step with accumulated grads + reset accumulators
                            steps = self.gradient_accumulation_steps
                            ...
                    else:
                        self._update_step(grads, trainable_variables)
                        # Run update step with accumulated grads + reset accumulators
                        steps = self.gradient_accumulation_steps
                        ...
                
```

def do_something_with(param):
return param2**

```

import re
import warnings

from keras.src import backend
from keras.src import initializers
from keras.src import optimizers
from keras.src import optimizers.schedules import learning_rate_schedule
from keras.src.saving import serialization_lib
from keras.src.saving.legacy import KerasSaveable
from keras.src.utils import tracking
from keras.src.utils import auto_name

class BaseOptimizer(KerasSaveable):
    def __init__(self,
                 learning_rate,
                 weight_decay=None,
                 clipnorm=None,
                 clipvalue=None,
                 global_clipnorm=None,
                 ema=None,
                 ema_momentum=0.99,
                 ema_overwrite_frequency=None,
                 loss_scale_factor=None,
                 gradient_accumulation_steps=None,
                 name=None,
                 **kwargs):
        self._lock = False
        if kwargs.pop("decay", None) is not None:
            warnings.warn("The argument 'decay' is no longer supported and will be ignored.")
        if kwargs:
            raise ValueError(f"Arguments not recognized: {list(kwargs)}")
        if name is None:
            name = auto_name(self, class_.__name__)
        self.name = name
        self.weight_decay = weight_decay
        self.clipnorm = clipnorm
        self.global_clipnorm = global_clipnorm
        self.clipvalue = clipvalue
        self.ema = ema
        self.loss_scale_factor = loss_scale_factor
        self.gradient_accumulation_steps = gradient_accumulation_steps
        if gradient_accumulation_steps:
            if not gradient_accumulation_steps == 2:
                raise ValueError(f"gradient_accumulation_steps must be an integer >= 2. "
                                 f"(Received: gradient_accumulation_steps={gradient_accumulation_steps})")
        dtype=reference_variable.dtype.
        name=name,
    )
    def _check_variables_are_known(self, variables):
        for v in variables:
            if self._var_key(v) not in self._trainable_variables_indices:
                raise RuntimeError(f"Variable {v} is not in the trainable_variables mapping. "
                                   f"This optimizer can only "
                                   f"be called for the variables it was originally built with. "
                                   f"When working with a new set of variables, you should "
                                   f"create a new optimizer instance.")
    def assign(self, variable, value):
        """Assign a value to a variable.
        This should be used in optimizers instead of `variable.assign(value)` to
        support backend specific optimizations.
        Note that the variable can be a model variable or an optimizer variable;
        it can be a backend native variable or a Keras variable.
        Args:
            variable: The variable to update.
            value: The value to add to the variable.
        variable.assign(value)
        """
        "Add a value to a variable.
        This should be used in optimizers instead of `variable.assign_add(value)` to
        support backend specific optimizations.
        Note that the variable can be a model variable or an optimizer variable;
        it can be a backend native variable or a Keras variable.
        Args:
            variable: The variable to update.
            value: The value to add to the variable.
        variable.assign_add(value)
        """
        "Add a value to a variable.
        This should be used in optimizers instead of `variable.assign_sub(value)` to
        support backend specific optimizations.
        Note that the variable can be a model variable or an optimizer variable;
        it can be a backend native variable or a Keras variable.
        Args:
            variable: The variable to update.
            value: The value to subtract from a variable.
        self._update_step(grad, variable, learning_rate):
            if isinstance(grad, Variable):
                self._track_variable(iterations)
                self._iterations = iterations
            # Create learning rate (schedule or variable)
            if isinstance(learning_rate, learning_rate_schedule.LearningRateSchedule):
                self._learning_rate = learning_rate.schedule(self._iterations)
            else:
                self._learning_rate = learning_rate
            self._learning_rate = learning_rate
            self._learning_rate = learning_rate
            if not isinstance(learning_rate, float):
                raise ValueError(
                    "The argument 'learning_rate' should be float, or an instance of "
                    "LearningRateSchedule, or a callable "
                    "(that takes in the current iteration value, "
                    "and returns the corresponding learning rate value). "
                    "(Received instead: learning_rate={learning_rate})")
            with backend.name_scope(self.name, caller=self):
                learning_rate = Backend.Variable(
                    learning_rate,
                    name="learning_rate",
                    dtype=backend.floatx(),
                    trainable=False,
                    aggregation="only_first_replica",
                    self._track_variable(learning_rate),
                    self._learning_rate = learning_rate
                )
            self._track_variable(self._learning_rate)
            self._tracker.add_to_store("variables", variable)
        @tracking.no_automatic_dependency_tracking
        def build(self, variables):
            if self.use_ema:
                for v in variables:
                    moving_average = []
                    if self.gradient_accumulation_steps:
                        self._accumulated_gradients = []
                    for i, v in enumerate(variables):
                        trainable_variables = v.trainable_variables
                        if self.use_ema:
                            self._moving_variables.append(v)
                            self._add_variable_to_reference(v, name="average")
                        moving_average.append(v)
                    self._grad_accumulated_gradients.append(moving_average)
            self._grad_accumulated_gradients.append(variable)
            self._add_variable_to_reference(variable, name="gradient_accumulator")
            self._trainable_variables = variables[]
            self._built = True
        def _var_key(self, variable):
            """Get a stable ID and the variable instance mapping.
            Returns:
                id(variable): (V) This optimizer can only "
                               "be called for the variables it was originally built with. "
                               "When working with a new set of variables, you should "
                               "create a new optimizer instance.
            """
            self._check_variables_are_known(variables)
            def get_variable_index(variable):
                return self._trainable_variables_indices[self._var_key(variable)]
            def add_variable(self,
                           shape,
                           initializer="zeros",
                           dtype=None,
                           aggregation="mean",
                           name=None,
                           self._check_super_called()):
                initializer = initializers.get(initializer)
                variable = Backend.Variable(
                    value=initializer.get(self.name, caller=self),
                    initializer=initializer,
                    shape=shape,
                    dtype=dtype,
                    trainable=False,
                    aggregation=aggregation,
                    name=name,
                    self._track_variable(variable)
                )
                self._trainable_variables.append(variable)
            def add_variable_from_reference(
                self,
                reference_variable,
                name=None,
                initializer="zeros"
            ):
                """Add an all-zeros variable with the shape and dtype of a reference
                variable.
                name = name + v.name
                if not reference_variable.path:
                    name = reference_variable.replace("\\", ".") + name
                else:
                    name = (
                        str(reference_variable.name).replace("\\", ".") + name
                        + "."
                    )
                return self._add_variable(
                    shape=reference_variable.shape,
                    name=name,
                    initializer=initializer
                )
            self._update_step(grads, trainable_variables):
                if not isinstance(grads, Variable):
                    raise NotImplementedError("grads should be a list of gradient tensors "
                                         "with 1:1 mapping to the list of variables the optimizer was built with."
                                         "trainable_variables can be provided on the first call to build the optimizer."
                                         "...")

                if len(grads) == 0:
                    # It is possible that the grad is empty. In this case,
                    # apply_gradients() is a no-op.
                    return

                if len(grads) != len(self._trainable_variables):
                    raise ValueError(
                        "When passing 'grads' as a list of gradient tensors, the "
                        "length of 'grads' must be equal to the length of 'variables'. "
                        "(Received a list of {len(grads)}) gradients, but the "
                        "optimizer is tracking {len(self._trainable_variables)} "
                        "trainable variables."
                    )

                if len(grads) != len(self._trainable_variables_indices):
                    raise ValueError(
                        "When passing 'grads' as a list of gradient tensors, the "
                        "length of 'grads' must be equal to the length of 'variables'. "
                        "(Received a list of {len(grads)}) gradients, but the "
                        "optimizer is tracking {len(self._trainable_variables)} "
                        "trainable variables."
                    )
                trainable_variables = self._trainable_variables
                grads = self._filter_empty_gradients(grads, trainable_variables)
                if len(list(grads)) == 0:
                    return

                if self._scale_gradients:
                    scale = self._loss_scale_factor
                    grads = [g / scale if g is not None else g / scale for g in grads]
                # Apply clipping.
                grads, trainable_variables = self._clip_gradients(grads, trainable_variables)
                self._apply_weight_decay(trainable_variables)

                # Apply gradient updates.
                self._backend_apply_gradients(grads, trainable_variables)
                # Apply gradient accumulation.
                for variable in trainable_variables:
                    if variable.constraint is not None:
                        variable.assign_variable.constraint(variable)

                self._backend_apply_gradients(self, grads, trainable_variables)
                # API method that can be overridden by different backends.
                JAX overrides it in order to deal with statelessness in gradient
                accumulation and EMA handling.

                The below implementation is intended to be generally backend-agnostic,
                but may not work with all backends.

                This method does 4 things:
                - Call the optimizer's update_step() to update trainable variables
                  and their gradients.
                - Update EMA variables, if EMA is configured.
                - Update gradient accumulators, if gradient accumulation is configured.
                - ...
                if self._gradient_accumulation_steps:
                    is_update_step = 1
                    if is_update_step % self._gradient_accumulation_steps == 0:
                        # % self._gradient_accumulation_steps == 0
                        # trainable_variables might have been filtered in previous
                        # processing steps, so we need to make sure the correct mapping between
                        # self._accumulated_gradients and trainable_variables
                        acc_grads = []
                        self._accumulated_gradients(self._get_variable_index(v))
                        for v in trainable_variables:
                            acc_grads.append(v)
                        self._update_step(np.concatenate(acc_grads), trainable_variables)
                    # Run update step with accumulated grads + reset accumulators
                    steps = self._gradient_accumulation_steps
                    ...
                self._update_step(grads, trainable_variables):
                    # Run update step with accumulated grads + reset accumulators
                    steps = self._gradient_accumulation_steps
                    ...
            
```

`def create_new_function(param):
do_something_with(param)`

`def reduce_dependency(param1, param2):
write_code_to_reduce_dependency(param1)
fine_tune_with(param2)`



DeepLearning.AI

Pair coding with an LLM

Transformers and code

Transformers in action

```
import re
import warnings

from keras import src.backend
from keras.src import initializers
from keras.src import ops
from keras.src.optimizers.schedules import learning_rate_schedule
from keras.src.saving import serialization_lib
from keras.src.saving.keras_saveable import KerasSaveable
from keras.src.utils import tracking
from keras.src.utils.naming import auto_name

import re
import warnings

from keras import src.backend
from keras.src import initializers
from keras.src import ops
from keras.src.optimizers.schedules import learning_rate_schedule
from keras.src.saving import serialization_lib
from keras.src.saving.keras_saveable import KerasSaveable
from keras.src.utils import tracking
from keras.src.utils.naming import auto_name

# Create learning rate (schedule or variable)
if instance(learning_rate, learning_rate_schedule.LearningRateSchedule):
    self._learning_rate = learning_rate
elif callable(learning_rate):
    self._learning_rate = learning_rate
else:
    if not isinstance(learning_rate, float):
        raise ValueError(
            "Argument `learning_rate` should be float, or an instance "
            "of LearningRateSchedule, or a callable "
            "representing a learning rate value. "
            "And returns the corresponding learning rate value."
            f"Received instead: learning_rate={learning_rate}")
    else:
        self._learning_rate = float(learning_rate)

# https://docs.google.com/presentation/d/1YuuiHYS3y4tbAbc0tCFhokmDJD_3FN847mgE0d2eww/
edit@slide-id:92e128fc34ee_0_162
with tracking.no_dependency_scope(self.name, caller=self):
    learning_rate = self._backend.Variable(
        learning_rate,
        name="learning_rate",
        aggregation="only_first_replica",
        trainable=False,
        aggregations="only_first_replica",
        )
    self._track_variable(learning_rate)
    self._learning_rate = learning_rate

def _track_variable(self, variable):
    self._tracker.add_to_store("variables", variable)

@tracking.no_automatic_dependency_tracking
def build(self, variables):
    if self.use_ema:
        self._trainable_variables["moving_average"] = []
    if self.gradient_accumulation_steps:
        self._accumulated_gradients = []
    for l, v in enumerate(variables):
        self._trainable_variables.setdefault(var_key(variable)) = l
    if self.use_ema:
        self._trainable_variables["moving_average"].append(
            self.add_variable_from_reference(
                variable,
                name="average",
            )
        )
    if self.gradient_accumulation_steps:
        self._accumulated_gradients.append(
            self.add_variable_from_reference(
                variable,
                name="gradient_accumulator",
            )
        )
    self._trainable_variables = variables[:]
    self.default = True

def var_key(self, variable):
    """To uniquely get a stable ID and the variable instance mapping.
    return id(variable)
    """
    def __hash__(self):
        return self._var_key(variable)

    @property
    def variables(self):
        return self._variables[:]

    def get_variable_index(self, variable):
        return self._trainable_variables_indices[self._var_key(variable)]
```

Transformers in action

```
import re
import warnings

from keras import src.backend
from keras.src import initializers
from keras.src import optimizers
from keras.src.optimizers.schedules import learning_rate_schedule
from keras.src.saving.keras_savable import KerasSavable
from keras.src.utils import tracking
from keras.src.utils.naming import auto_name

import re
import warnings

from keras import src.backend
from keras.src import initializers
from keras.src import optimizers
from keras.src.optimizers.schedules import learning_rate_schedule
from keras.src.saving.keras_savable import KerasSavable
from keras.src.utils import tracking
from keras.src.utils.naming import auto_name

# Create learning rate (schedule or variable)
if isinstance(learning_rate, learning_rate_schedule.LearningRateSchedule):
    self._learning_rate = learning_rate
elif callable(learning_rate):
    self._callables[learning_rate] = learning_rate
else:
    if not isinstance(learning_rate, float):
        raise ValueError(
            "Argument `learning_rate` should be float, or an instance "
            "of LearningRateSchedule, or a callable "
            "that takes a float and returns the corresponding learning rate value."
            "\nReceived instead: learning_rate={learning_rate}")
    else:
        self._learning_rate = float(learning_rate)

# Track variable(learning_rate)
self._track_variable(learning_rate)
self._learning_rate = learning_rate

def track_variable(self, variable):
    self._tracker.add_to_store("variables", variable)

@tracking.no_automatic_dependency_tracking
def build(self, variables):
    if self._use_ema:
        self._trainable_variables["moving_average"] = []
    if self._gradient_accumulation_steps:
        self._accumulated_gradients = []
    for l, v in enumerate(variables):
        if v in self._trainable_variables:
            self._trainable_variables[l][var_key(variable)] = v
        if self._use_ema:
            self._trainable_variables["moving_average"].append(
                self._add_variable_from_reference(
                    variable,
                    name="average",
                )
            )
        if self._gradient_accumulation_steps:
            self._accumulated_gradients.append(
                self._add_variable_from_reference(
                    variable,
                    name="gradient_accumulator",
                )
            )
    self._trainable_variables = variables[:]
    self._default = True

def var_key(self, variable):
    # To make sure we get a stable ID and the variable instance mapping.
    return id(variable)

@property
def variables(self):
    return self._variables[:]

def get_variable_index(self, variable):
    return self._trainable_variables_indices[variable]
```

You have a syntax error here: this should be:

from keras.src import backend

You have an error in your class initialization: the keyword clifnorm is not recognized.

The use of a for loop here is inefficient. Consider using a vectorized approach instead.

Updating code

```
import tensorflow as tf  
import tensorflow.keras as tfk
```

```
def create_new_function(param):  
    do_something_with(param)
```

```
def reduce_dependency(param1, param2):  
    write_code_to_reduce_dependency(param1)  
    fine_tune_with(param2)
```



Update! Just use 'import keras as k'

Writing documentation

```
def create_new_function(param):  
    do_something_with(param)
```

```
def reduce_dependency(param1, param2):  
    write_code_to_reduce_dependency(param1)  
    fine_tune_with(param2)
```

```
# This function does something really important  
def create_new_function(param):  
    do_something_with(param)
```

```
# This function is even more important  
def reduce_dependency(param1, param2):  
    # We all love to reduce dependencies  
    write_code_to_reduce_dependency(param1)  
    # Oh and that tuning is just fine  
    fine_tune_with(param2)
```

Dealing with technical debt

```
import re
import warnings

from keras.src import backend
from keras.src import initializers
from keras.src import optimizers
from keras.src import schedules
from keras.src.saving.keras_serialization import learning_rate_schedule
from keras.src.saving.keras_savable import KerasSavable
from keras.src.utils import tracking
from keras.src.utils.naming import auto_name

import BaseOptimizer(KerasSavable):
    def __init__(_
        self,
        learning_rate,
        weight_decay=None,
        clipnorm=None,
        clipvalue=None,
        global_clipnorm=None,
        use_ema=False,
        ema_momentum=0.99,
        ema_overwrite_frequency=None,
        loss_scale_factor=None,
        gradient_accumulation_steps=None,
        name=None,
        *kwargs,
    ):
        self._lock = False

        if keras.optimizers.decay is not None:
            warnings.warn(
                "Argument `decay` is no longer supported and will be ignored."
            )
        kwargs.pop("decay", None)
        if "decay" in kwargs:
            raise ValueError(f"Arguments(s) not recognized: {kwargs}")

        if name is None:
            name = auto_name(self, class_=__name__)
        self.name = name
        self.weight_decay = weight_decay
        self.clipnorm = clipnorm
        self.global_clipnorm = global_clipnorm
        self.clipvalue = clipvalue
        self.loss_scale = loss_scale
        self.loss_scale_factor = loss_scale_factor
        self.gradient_accumulation_steps = gradient_accumulation_steps

        if gradient_accumulation_steps:
            if not gradient_accumulation_steps >= 2:
                raise ValueError(
                    f"gradient_accumulation_steps must be an integer >= 2. "
                    f"Received: gradient_accumulation_steps={gradient_accumulation_steps}"
                )
        dtype=reference_variable.dtype,
        name=name,
    )

    def _check_variables_are_known(self, variables):
        for v in variables:
            if self._var_key(v) not in self._trainable_variables_indices:
                raise ValueError(f"Variable {v} not found. This optimizer was built with "
                                f"the assumption that all variables are known. "
                                f"When working with a new set of variables, you should "
                                f"recreate a new optimizer instance."
    )

    # Create learning rate (schedule or variable)
    if isinstance(learning_rate, learning_rate_schedule.LearningRateSchedule):
        self._learning_rate = learning_rate
    else:
        self._learning_rate = learning_rate
    self._learning_rate_variable = self._learning_rate
    else:
        if not isinstance(learning_rate, float):
            raise ValueError(
                f"Argument `learning_rate` should be float, or an instance "
                f"of LearningRateSchedule. Received: {learning_rate}"
            )
        self._learning_rate = float(learning_rate)

    with backend.name_scope(self.name, caller=self):
        learning_rate = backend.Variable(
            learning_rate,
            name="learning_rate",
            dtype=backend.float32,
            trainable=False,
            aggregation="only_first_replica"
        )
        self._track_variable(learning_rate)
        self._learning_rate = learning_rate

    def _track_variable(self, variable):
        self._tracker.add_to_store("variables", variable)

    @tracking.no_automatic_dependency_tracking
    def build(self, variables):
        if self._lock:
            raise RuntimeError("Optimizer has been built and cannot be modified")
        self._model_variables_moving_average = []
        if self._gradient_accumulation_steps:
            self._gradient_accumulation_steps.append(0)
        for i, variable in enumerate(variables):
            self._trainable_variables_indices[self._var_key(variable)] = i
            if self._model_variables_moving_average:
                self._model_variables_moving_average.append(
                    self._add_variable_from_reference(
                        variable,
                        name="average",
                    )
                )
        if self._gradient_accumulation_steps:
            self._accumulated_gradients.append(
                self._add_variable_from_reference(
                    variable,
                    name="gradient_accumulator",
                )
            )
        self._trainable_variables = variables[:]
        self._built = True

    def var_key(self, variable):
        # If var function to get a stable ID and the variable in state.
        return id(variable)

    def variables(self):
        return self._variables[:]

    def _get_trainable_variables_indices(self, variable):
        return self._trainable_variables_indices[self._var_key(variable)]
```

This piece of code does foobarbaz. Should update to version 10.1, and then memory leaks will be avoided with:

`def newbarbaz():`

`do_newbarbaz()`

This function is deprecated, please update per <http://docs.com/deprecatedfunction>





DeepLearning.AI

Pair coding with an LLM

Generating code with chatbots

Three ways of asking an LLM to add numbers

Can you write a Python function to add two numbers named 'a' and 'b' and return the result?

Three ways of asking an LLM to add numbers

Can you write a Python function to add two numbers named 'a' and 'b' and return the result?

Create a JavaScript function that adds two numbers.

Three ways of asking an LLM to add numbers

Can you write a Python function to add two numbers named 'a' and 'b' and return the result?

Create a JavaScript function that adds two numbers.

Write a C# method to add two numbers.

Three ways of asking an LLM to add numbers

Can you write a Python function to add two numbers named 'a' and 'b' and return the result?

Create a JavaScript function that adds two numbers.

Write a C# **method** to add two numbers.

Debugging code with an LLM

```
def calculate_average(numbers):
    total_sum = sum(numbers)
    count = len(numbers)
    average = total_sum / count

    return average
```

Debugging code with an LLM

```
def calculate_average(numbers):
    total_sum = sum(numbers)
    count = len(numbers)
    average = total_sum / count
    return average
```

An example of an effective prompt

Please write a Python function named `calculate_area` that takes an argument `radius`.

The function should calculate the area of a circle given the radius.

Ensure that the function handles non-numeric inputs by raising a `ValueError` with the message ‘Input must be a numeric value.’

Include comments in the code explaining each step.

An example of an effective prompt

Prompt Please write a Python function named calculate_area that takes an argument radius.

The function should calculate the area of a circle given the radius.
Ensure that the function handles non-numeric inputs by raising a ValueError with the message 'Input must be a numeric value.'

Include comments in the code explaining each step.



DeepLearning.AI

Pair coding with an LLM

Iterative prompting

Context in prompting

Context is information you provide in your prompt to steer the LLM to a better answer.

Adding context can dramatically change the model's output.



DeepLearning.AI

Pair coding with an LLM

Giving the LLM feedback

Palindromes

Strings that read the same forwards as backwards

A man, a plan, a canal, Panama



DeepLearning.AI

Pair coding with an LLM

Assigning the LLM a role

```
if (number > 5) {  
    result = "Number is greater than 5";  
} else {  
    result = "Number is 5 or less";  
}
```

```
if (number > 5) {  
    result = "Number is greater than 5";  
} else {  
    result = "Number is 5 or less";  
}
```

```
result = (number > 5) ? "Number is greater than 5" : "Number is 5 or less";
```

Prompts with and without roles

Prompt 1: Write a Python function to calculate a factorial.

Prompt 2: As my Python mentor, please write a function to calculate a factorial and explain how it works.

Prompt 1: Write a Python function to calculate a factorial.

Response:



Here's a Python function to calculate the factorial of a given number:

python

Copy

```
def factorial(n):
    """Calculate the factorial of a non-negative integer n."""
    if n < 0:
        raise ValueError("Factorial is not defined for negative numbers")
    elif n == 0 or n == 1:
        return 1
    else:
        result = 1
        for i in range(2, n + 1):
            result *= i
        return result

# Example usage:
print(factorial(5)) # Output: 120
```

This function uses an iterative approach to calculate the factorial. It raises an error if the input is a negative number, returns 1 if the input is 0 or 1, and calculates the factorial for other positive integers using a loop.

Basic Prompt

Prompt: Explain how to create a list in Python and add elements to it.

Prompt with role assignment

Prompt: As a beginner Python tutor, explain how to create a list in Python and add elements to it.

Prompt with role assignment

Prompt: As a beginner Python tutor, explain how to create a list in Python and add elements to it.

Prompt with role assignment

Prompt: As a beginner Python tutor, explain how to create a list in Python and add elements to it.

Alternative Prompt: You are a friendly coding mentor. Explain how to create a list in Python and add elements to it.

Prompt with role assignment

Prompt: As a friendly code guide, please explain how loops work in Python.



DeepLearning.AI

Pair coding with an LLM

Leveling up with multiple roles

Assigning multiple roles

Prompt: As both a software architect and a security expert, evaluate this Python script for a web application and suggest architectural improvements and security enhancements.

Assigning multiple roles

Prompt: As both a **software architect** and a **security expert**, evaluate this Python script for a web application and suggest architectural improvements and security enhancements.

Assigning multiple roles

Prompt: As both a **software architect** and a **security expert**, evaluate this Python script for a web application and suggest architectural improvements and security enhancements.

```
def storeuserdata(user_data):
    database = open('user_database.txt', 'a')
    database.write(str(user_data))
    database.close()

storeuserdata({'username': 'admin', 'password': '1234'})
```

How would humans respond?

Software architect

Assess code structure:

- Is code scalable?
- Any redundancies impacting performance?
- Is software design suitable for the task?

Security Expert

Look for vulnerabilities:

- Insecure data handling?
- Exposure to SQL injection?
- Exposure to cross-scripting attacks

Assigning multiple roles

Prompt: As a contributor to open-source Python projects, critique this Python library for data visualization and suggest what features could make it comparable to major existing libraries like Matplotlib or Seaborn

```
class SimplePlot:  
    def init(self, data):  
        self.data = data  
  
    def line_plot(self):  
        plt.plot(self.data)  
        plt.show()  
  
Example usage  
plot = SimplePlot([1, 2, 3, 4, 5])  
plot.line_plot()
```

Assigning multiple roles

Prompt: As a contributor to open-source Python projects, critique this Python library for data visualization and suggest what features could make it comparable to major existing libraries like Matplotlib or Seaborn

As a contributor to open-source Python projects, critique this Python library for data visualization and suggest what features could make it comparable to major existing libraries like Matplotlib or Seaborn

As a contributor to open-source Python projects, critique this Python library for data visualization and suggest what features could make it comparable to major existing libraries like Matplotlib or Seaborn

```
class SimplePlot:  
    def init(self, data):  
        self.data = data  
  
    def line_plot(self):  
        plt.plot(self.data)  
        plt.show()
```

Example usage

```
plot = SimplePlot([1, 2, 3, 4, 5])  
plot.line_plot()
```



DeepLearning.AI

Pair coding with an LLM

Expert roles for
specialized knowledge

Python library critique

```
import matplotlib.pyplot as plt
class DataVisualizer:
    def init(self, data):
        self.data = data
    def plot(self, kind='line'):
        if kind == 'line':
            plt.plot(self.data)
        elif kind == 'bar':
            plt.bar(range(len(self.data)), self.data)
        plt.show()
    # Example usage
visualizer = DataVisualizer([10, 20, 30, 40, 50])
visualizer.plot('bar')
```

Export role: open source contributor

Prompt As a contributor to open-source Python projects, critique this Python library for data visualization and suggest enhancements to make it comparable to major libraries like Matplotlib or Seaborn.

```
import matplotlib.pyplot as plt
class DataVisualizer:
    def init(self, data):
        self.data = data
    def plot(self, kind='line'):
        if kind == 'line':
            plt.plot(self.data)
        elif kind == 'bar':
            plt.bar(range(len(self.data)), self.data)
        plt.show()
# Example usage
visualizer = DataVisualizer([10, 20, 30, 40, 50])
visualizer.plot('bar')
```

Adding NLP features

```
import nltk
class TextAnalyzer:
    def init(self, text):
        self.text = text
    def summarize(self):
        sentences = nltk.sent_tokenize(self.text)
        return ' '.join(sentences[:2])
```

Export role: NLP expert

Prompt As an NLP expert, suggest improvements to this text summarization feature to enhance its functionality and accuracy.

```
import nltk
class TextAnalyzer:
    def init(self, text):
        self.text = text
    def summarize(self):
        sentences = nltk.sent_tokenize(self.text)
        return ' '.join(sentences[:2])
```

Finding edge cases

```
def calculatediscount(price, discount):
    if discount > 100:
        raise ValueError('Discount cannot exceed 100%')
    return price * (100 - discount) / 100
# Test cases
print(calculatediscount(100, 105)) # Should raise an exception
```

Export role: Software tester

Prompt As a software tester, analyze this function for potential edge cases and suggest robust handling strategies.

```
def calculatediscount(price, discount):
    if discount > 100:
        raise ValueError('Discount cannot exceed 100%')
    return price * (100 - discount) / 100
# Test cases
print(calculatediscount(100, 105)) # Should raise an exception
```

Prompting Techniques

- **Be specific:** Provide detail and context about your problem
- **Assign a Role:** Assign a role to tailor the output you receive
- **Request an Expert Opinion:** Assign an expert role and ask the LLM to evaluate the work you've already done to further refine it
- **Give Feedback:** Iteratively prompt the LLM and provide feedback on the output you receive to get closer to your expected results

LLM Best Practices

- Experiment with what your LLM can do
- Test LLM code carefully
- Use LLMs as a learning tool
- You are the context expert



DeepLearning.AI

Pair coding with an LLM

Best practices

Best practices

- **Be specific:** Provide detail and context about your problem
- **Assign a role:** Assign a role to tailor the output you receive
- **Request an expert opinion:** Assign an expert role and ask the LLM to evaluate the work you've already done to further refine it
- **Give feedback:** Iteratively prompt the LLM and provide feedback on the output you receive to get closer to your expected results

Getting the most out of LLMs

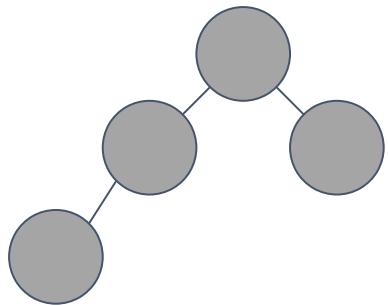
- Experiment with what your LLM can do
- Test LLM code carefully
- Use LLMs as a learning tool
- You are the context expert



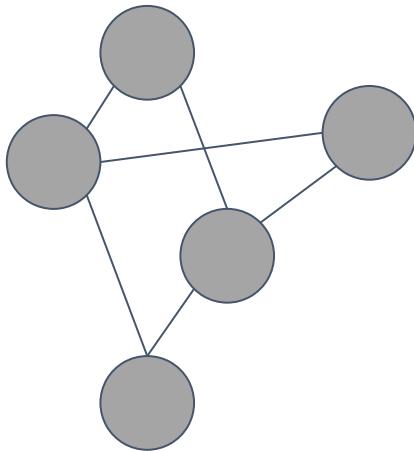
DeepLearning.AI

Leveraging LLMs for code analysis

Data structures in
production



Tree



Graph

Key 1		Values
Key 2		Values
Key 3		Values
...	
Key N		Values

Hash Table



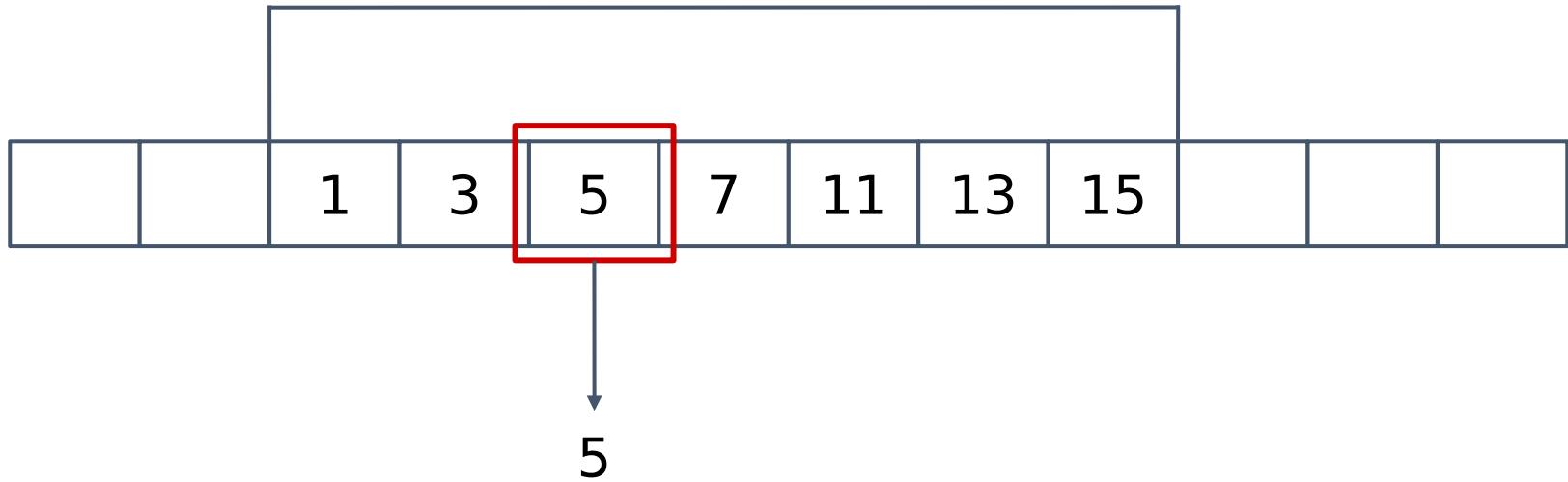
DeepLearning.AI

Leveraging LLMs for code analysis

Arrays

Arrays

Memory allocated



```
# Creating an array in Python
import array as arr
a = arr.array('i', [1, 2, 3, 4, 5]) # 'i' is the type code for integers

# Accessing elements
print("First element:", a[0])
print("Second element:", a[1])

# Modifying an element
a[2] = 6
print("Modified array:", a)
```

Analyzing the performance of arrays

Prompt: What would happen if I had nothing in my array?

Analyzing the performance of arrays

- Prompts:** What would happen if I had nothing in my array?
- Follow up:** What would happen if I had billions of numbers in my array?

Analyzing the performance of arrays

Prompt: What risks would I have implementing billions of numbers using an array?

Analyzing the performance of arrays

Prompt: Take on the role of an expert software engineer, what advice would you give me in using an array like this?

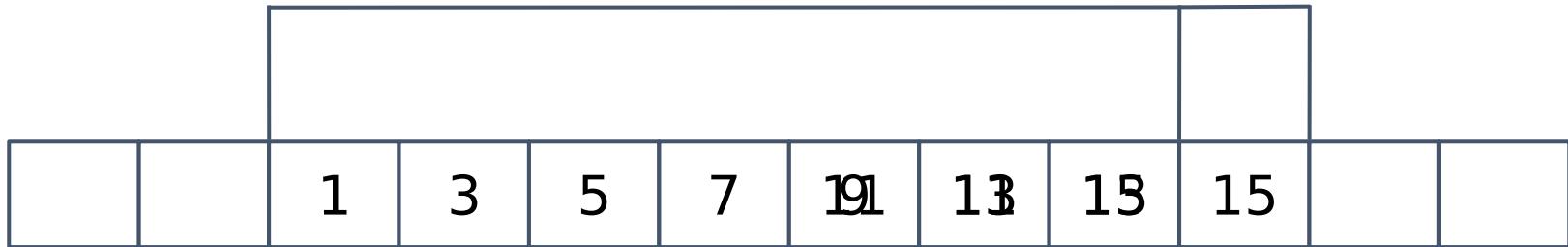
Limitations of Arrays

Memory allocated



Limitations of Arrays

Memory allocated

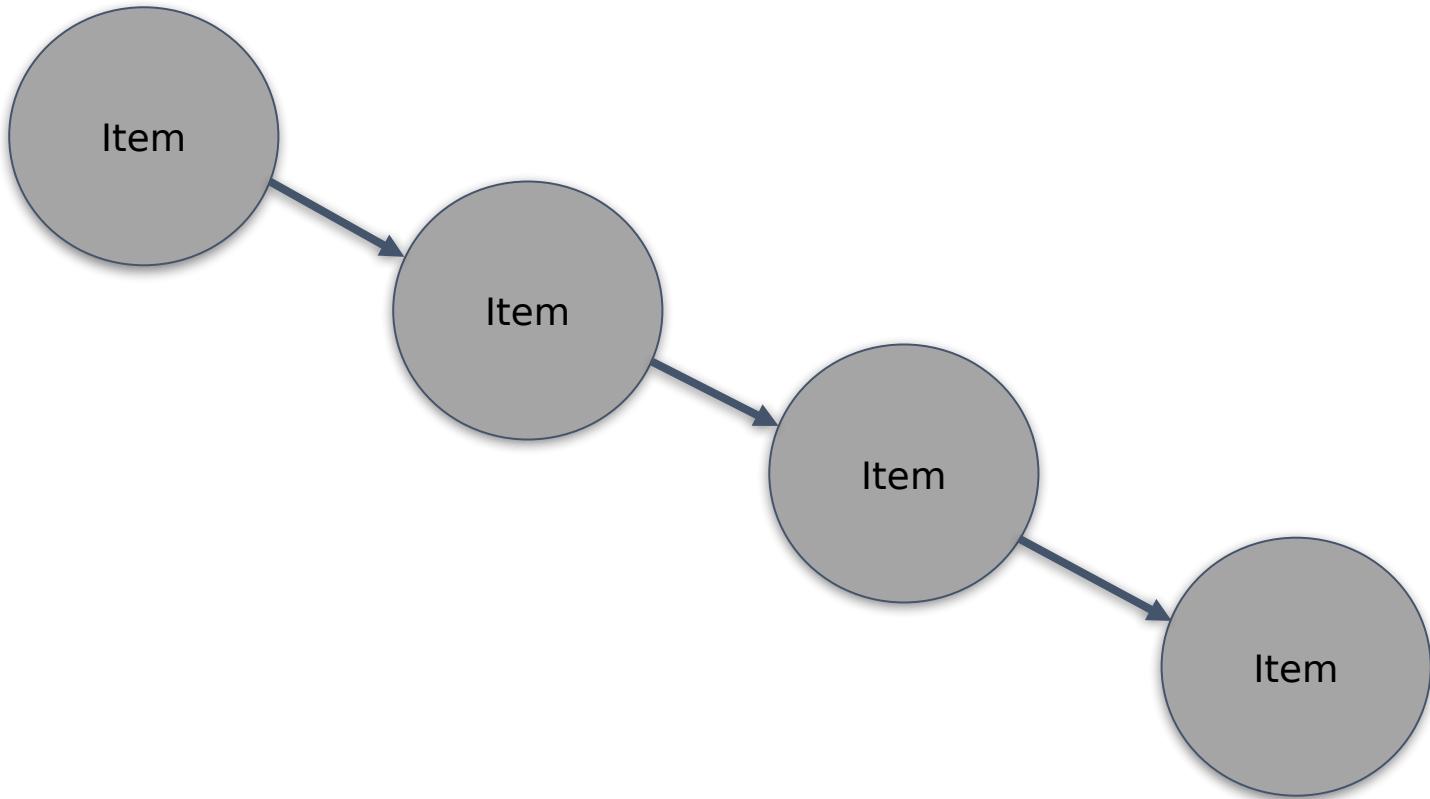


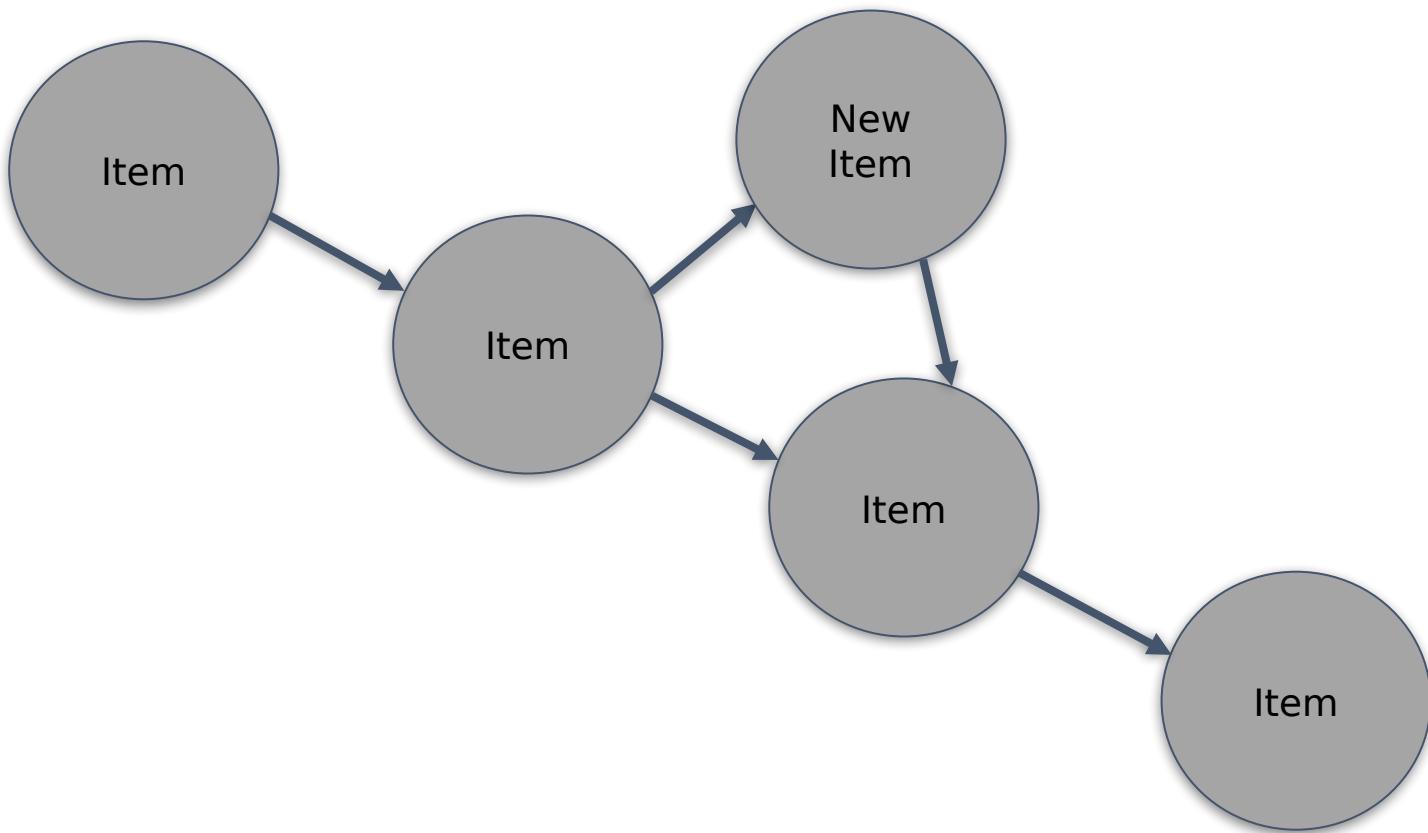


DeepLearning.AI

Leveraging LLMs for code analysis

Linked lists





Analyzing the performance of linked lists

Prompts: Write some code to implement a linked list in python.

Analyzing the performance of linked lists

Prompts: Write some code to implement a linked list in python.

What are the downsides and overheads associated with linked lists compared to other python data structures?



Using a linked list data structure like the one provided has several overheads and downsides compared to other data structures like arrays or Python lists. Here are some considerations:

1. **Memory Overhead:** Each element in the linked list requires additional memory to store the reference to the next node (`self.next`). This overhead can be significant for large linked lists, especially if the data stored in each node is small.
2. **Random Access:** Unlike arrays or Python lists, linked lists do not support constant-time random access to elements. Accessing an element at a specific index requires traversing the list from the beginning, which results in a time complexity of $O(n)$, where n is the number of elements in the list.
3. **Cache Performance:** Linked lists can exhibit poor cache performance compared to contiguous data structures like arrays. This is because elements in a linked list are stored in scattered memory locations, which may result in more cache misses during traversal.
4. **Extra Pointer Overhead:** In addition to the data stored in each node, linked lists require extra memory to store pointers or references to connect the nodes. This overhead can become significant for large lists and can affect memory usage and performance.
5. **Insertion and Deletion Complexity:** While insertion and deletion operations can be efficient for singly linked lists ($O(1)$ for insertion/deletion at the beginning, $O(n)$ for insertion/deletion at the end or in the middle), they still involve pointer manipulation and traversal, which can be slower compared to arrays.
6. **Sequential Access:** Although traversal of a linked list is straightforward, sequential access can still be slower compared to arrays due to the lack of locality of reference. In arrays, elements are stored contiguously in memory, which can lead to better performance for sequential access patterns.
7. **Additional Overhead for Doubly Linked Lists:** If you need bidirectional traversal, you might opt for a doubly linked list, where each node has a reference to both the next and previous nodes. However, this comes with additional overhead for storing the previous node reference in each node.

Despite these overheads and downsides, linked lists have their own advantages, such as efficient insertion and deletion at the beginning (for singly linked lists), dynamic memory allocation, and flexibility in size. They are particularly useful in scenarios where frequent insertion and deletion operations are required, and the size of the data structure varies dynamically.



Using a linked list data structure like the one provided has several overheads and downsides compared to other data structures like arrays or Python lists. Here are some considerations:

1. **Memory Overhead:** Each element in the linked list requires additional memory to store the reference to the next node (`self.next`). This overhead can be significant for large linked lists, especially if the data stored in each node is small.
2. **Random Access:** Unlike arrays or Python lists, linked lists do not support constant-time random access to elements. Accessing an element at a specific index requires traversing the list from the beginning, which results in a time complexity of $O(n)$, where n is the number of elements in the list.
3. **Cache Performance:** Linked lists can exhibit poor cache performance compared to contiguous data structures like arrays. This is because elements in a linked list are stored in scattered memory locations, which may result in more cache misses during traversal.
4. **Extra Pointer Overhead:** In addition to the data stored in each node, linked lists require extra memory to store pointers or references to connect the nodes. This overhead can become significant for large lists and can affect memory usage and performance.
5. **Insertion and Deletion Complexity:** While insertion and deletion operations can be efficient for singly linked lists ($O(1)$ for insertion/deletion at the beginning, $O(n)$ for insertion/deletion at the end or in the middle), they still involve pointer manipulation and traversal, which can be slower compared to arrays.
6. **Sequential Access:** Although traversal of a linked list is straightforward, sequential access can still be slower compared to arrays due to the lack of locality of reference. In arrays, elements are stored contiguously in memory, which can lead to better performance for sequential access patterns.
7. **Additional Overhead for Doubly Linked Lists:** If you need bidirectional traversal, you might opt for a doubly linked list, where each node has a reference to both the next and previous nodes. However, this comes with additional overhead for storing the previous node reference in each node.

Despite these overheads and downsides, linked lists have their own advantages, such as efficient insertion and deletion at the beginning (for singly linked lists), dynamic memory allocation, and flexibility in size. They are particularly useful in scenarios where frequent insertion and deletion operations are required, and the size of the data structure varies dynamically.

Asking clarifying questions

Prompt: Can you give me an example of where (3) could be a problem?

Can you give me an example of where (3) could be a problem?



Certainly! Let's consider a scenario where cache performance becomes a significant concern when using a linked list: iterating over a large linked list.

Imagine you have a large dataset stored in a linked list, and you need to perform some operation on each element of the list. This operation involves accessing the data of each node sequentially.

Due to the scattered memory locations of the nodes in the linked list, accessing each node's data may result in cache misses. When a cache miss occurs, the CPU has to fetch the data from the main memory, which is much slower compared to accessing data from the cache.

Here's an example in Python:

python

Copy

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

# Create a large linked list with 10 million nodes
def create_large_linked_list():
    head = Node(0)
    current = head
    for i in range(1, 10000000):
        current.next = Node(i)
        current = current.next
    return head
```

Analyzing the performance of linked lists

Prompt: Take on the role of an expert software developer at a company that suffers from denial of service attacks. If I implement some routines with code like this, what risks am I facing?

Analyzing the performance of linked lists

Prompt: How would you modify the code to mitigate against these?

```

import threading

class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class LinkedList:
    def __init__(self, max_size=None):
        self.head = None
        self.size = 0
        self.max_size = max_size
        self.lock = threading.Lock()

    def append(self, data):
        # Validate input data
        if len(data) > 1000:
            raise ValueError("Data size exceeds maximum limit")

    with self.lock:
        if self.max_size is not None and self.size >= self.max_size:
            raise ValueError("Linked list is full")
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
        else:
            last = self.head
            while last.next:
                last = last.next
            last.next = new_node
            self.size += 1

    def print_list(self):
        current = self.head
        while current:
            print(current.data, end=" ")
            current = current.next

```

1. Add code to remove a node

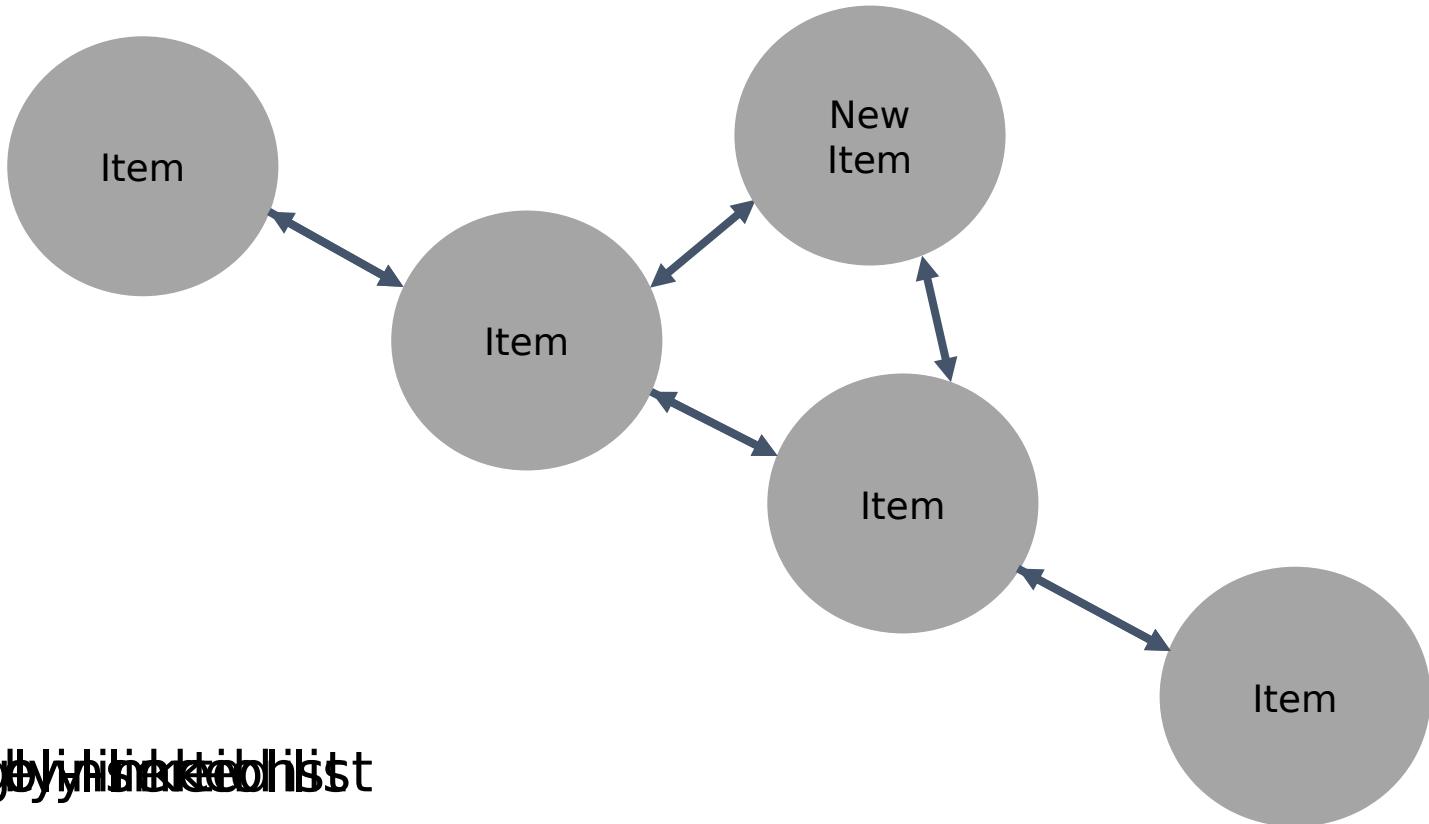
2. Have LLM profile new code



DeepLearning.AI

Leveraging LLMs for code analysis

Self-paced exercise:
Doubly-linked lists



~~Single linked list~~

Benefits of doubly-linked lists

Prompt: Why would I want to implement a doubly-linked list instead of a singly-linked one?



Using a doubly linked list instead of a singly linked list can be beneficial in several scenarios:

1. **Bidirectional Traversal:** Doubly linked lists allow traversal both forward and backward, as each node contains references to both its previous and next nodes. If your application frequently needs to traverse the list in both directions, a doubly linked list can provide better performance compared to a singly linked list, where backward traversal requires iterating from the head of the list each time.
2. **Insertions and Deletions at Arbitrary Positions:** Doubly linked lists are more efficient for insertions and deletions at arbitrary positions within the list. In a singly linked list, to delete a node, you typically need to traverse the list from the head to find the node to delete and its predecessor, which has a time complexity of $O(n)$. With a doubly linked list, you can delete a node in constant time if you already have a reference to it.
3. **Implementation of Data Structures:** Doubly linked lists are often used as the underlying data structure for other abstract data types, such as deques (double-ended queues) and queues, where efficient insertion and deletion operations at both ends of the list are required.
4. **Memory Overhead:** Doubly linked lists require additional memory for storing references to both the next and previous nodes compared to singly linked lists. However, in many scenarios, the benefits of bidirectional traversal and efficient insertions/deletions outweigh the additional memory overhead.
5. **Caching and Memory Management:** Doubly linked lists can be beneficial in scenarios where caching or memory management is a concern. For example, in certain caching algorithms like LRU (Least Recently Used), a doubly linked list can efficiently track the order of elements based on their access time.

In summary, consider using a doubly linked list when bidirectional traversal, efficient insertions/deletions at arbitrary positions, or specialized data structures built on top of linked lists are required for your application. However, be mindful of the additional memory overhead associated with doubly linked lists.



Using a doubly linked list instead of a singly linked list can be beneficial in several scenarios:

1. **Bidirectional Traversal:** Doubly linked lists allow traversal both forward and backward, as each node contains references to both its previous and next nodes. If your application frequently needs to traverse the list in both directions, a doubly linked list can provide better performance compared to a singly linked list, where backward traversal requires iterating from the head of the list each time.
2. **Insertions and Deletions at Arbitrary Positions:** Doubly linked lists are more efficient for insertions and deletions at arbitrary positions within the list. In a singly linked list, to delete a node, you typically need to traverse the list from the head to find the node to delete and its predecessor, which has a time complexity of $O(n)$. With a doubly linked list, you can delete a node in constant time if you already have a reference to it.
3. **Implementation of Data Structures:** Doubly linked lists are often used as the underlying data structure for other abstract data types, such as deques (double-ended queues) and queues, where efficient insertion and deletion operations at both ends of the list are required.
4. **Memory Overhead:** Doubly linked lists require additional memory for storing references to both the next and previous nodes compared to singly linked lists. However, in many scenarios, the benefits of bidirectional traversal and efficient insertions/deletions outweigh the additional memory overhead.
5. **Caching and Memory Management:** Doubly linked lists can be beneficial in scenarios where caching or memory management is a concern. For example, in certain caching algorithms like LRU (Least Recently Used), a doubly linked list can efficiently track the order of elements based on their access time.

In summary, consider using a doubly linked list when bidirectional traversal, efficient insertions/deletions at arbitrary positions, or specialized data structures built on top of linked lists are required for your application. However, be mindful of the additional memory overhead associated with doubly linked lists.

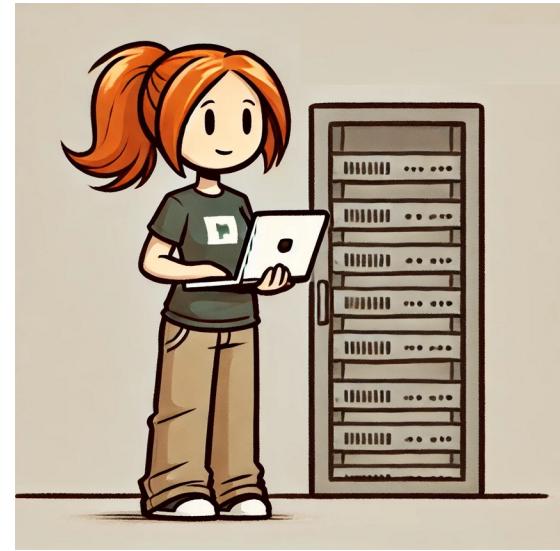


Using a doubly linked list instead of a singly linked list can be beneficial in several scenarios:

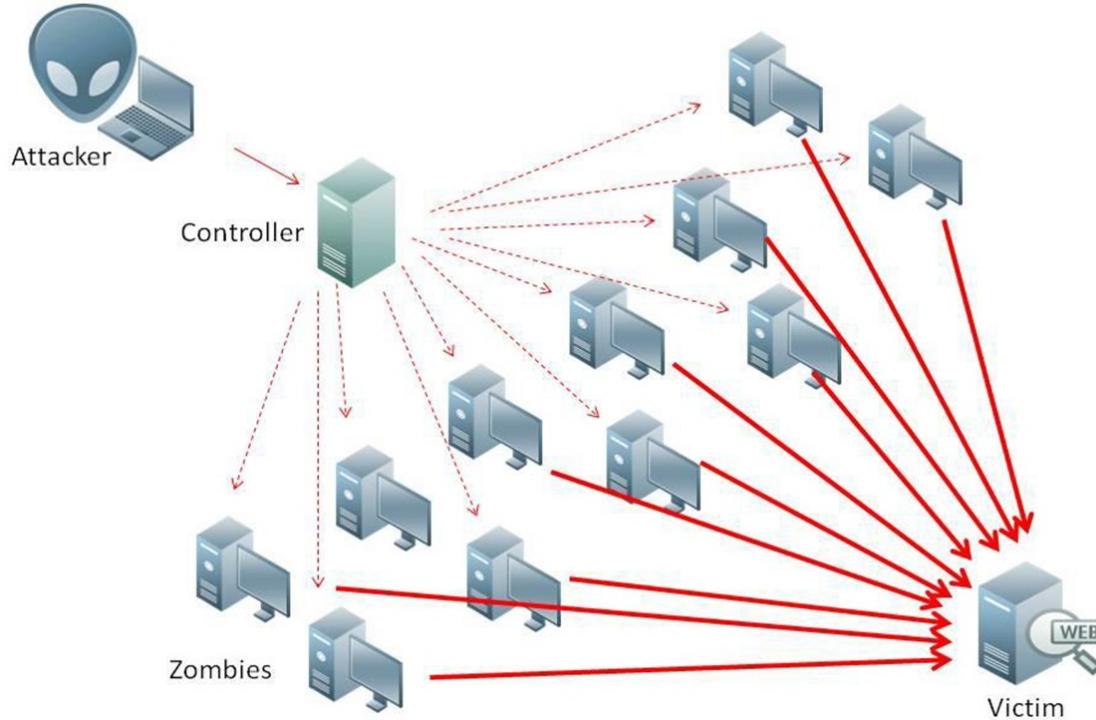
1. **Bidirectional Traversal:** Doubly linked lists allow traversal both forward and backward, as each node contains references to both its previous and next nodes. If your application frequently needs to traverse the list in both directions, a doubly linked list can provide better performance compared to a singly linked list, where backward traversal requires iterating from the head of the list each time.
2. **Insertions and Deletions at Arbitrary Positions:** Doubly linked lists are more efficient for insertions and deletions at arbitrary positions within the list. In a singly linked list, to delete a node, you typically need to traverse the list from the head to find the node to delete and its predecessor, which has a time complexity of $O(n)$. With a doubly linked list, you can delete a node in constant time if you already have a reference to it.
3. **Implementation of Data Structures:** Doubly linked lists are often used as the underlying data structure for other abstract data types, such as deques (double-ended queues) and queues, where efficient insertion and deletion operations at both ends of the list are required.
4. **Memory Overhead:** Doubly linked lists require additional memory for storing references to both the next and previous nodes compared to singly linked lists. However, in many scenarios, the benefits of bidirectional traversal and efficient insertions/deletions outweigh the additional memory overhead.
5. **Caching and Memory Management:** Doubly linked lists can be beneficial in scenarios where caching or memory management is a concern. For example, in certain caching algorithms like LRU (Least Recently Used), a doubly linked list can efficiently track the order of elements based on their access time.

In summary, consider using a doubly linked list when bidirectional traversal, efficient insertions/deletions at arbitrary positions, or specialized data structures built on top of linked lists are required for your application. However, be mindful of the additional memory overhead associated with doubly linked lists.

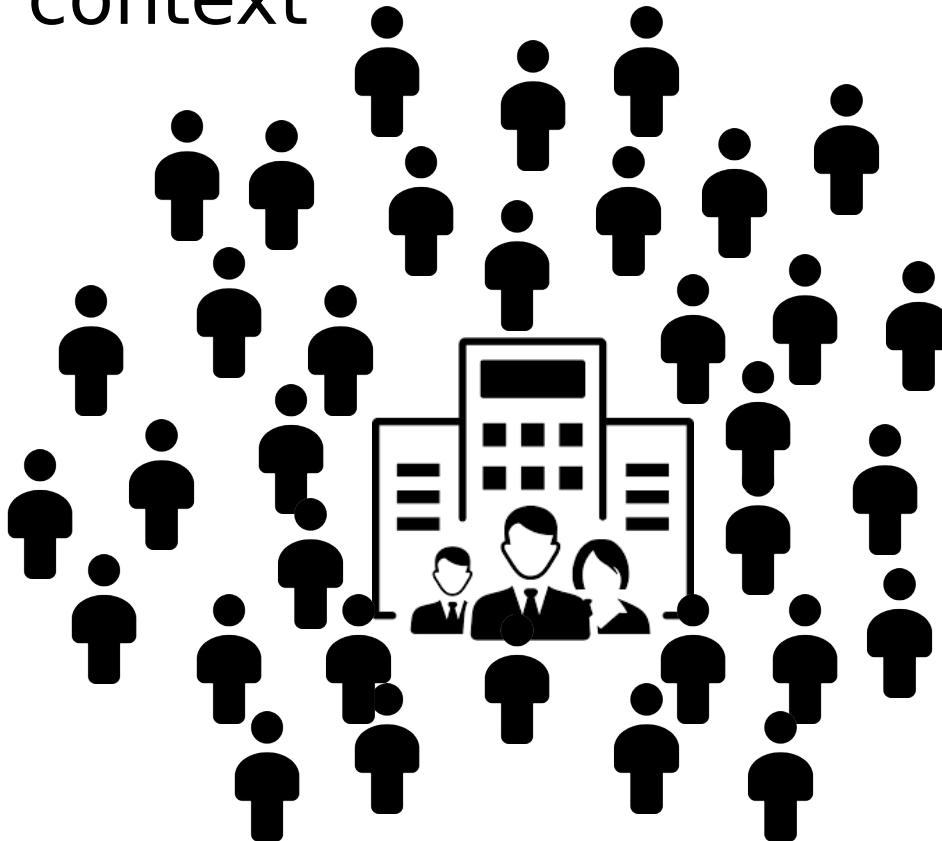
Role assignment



Company context



Company context

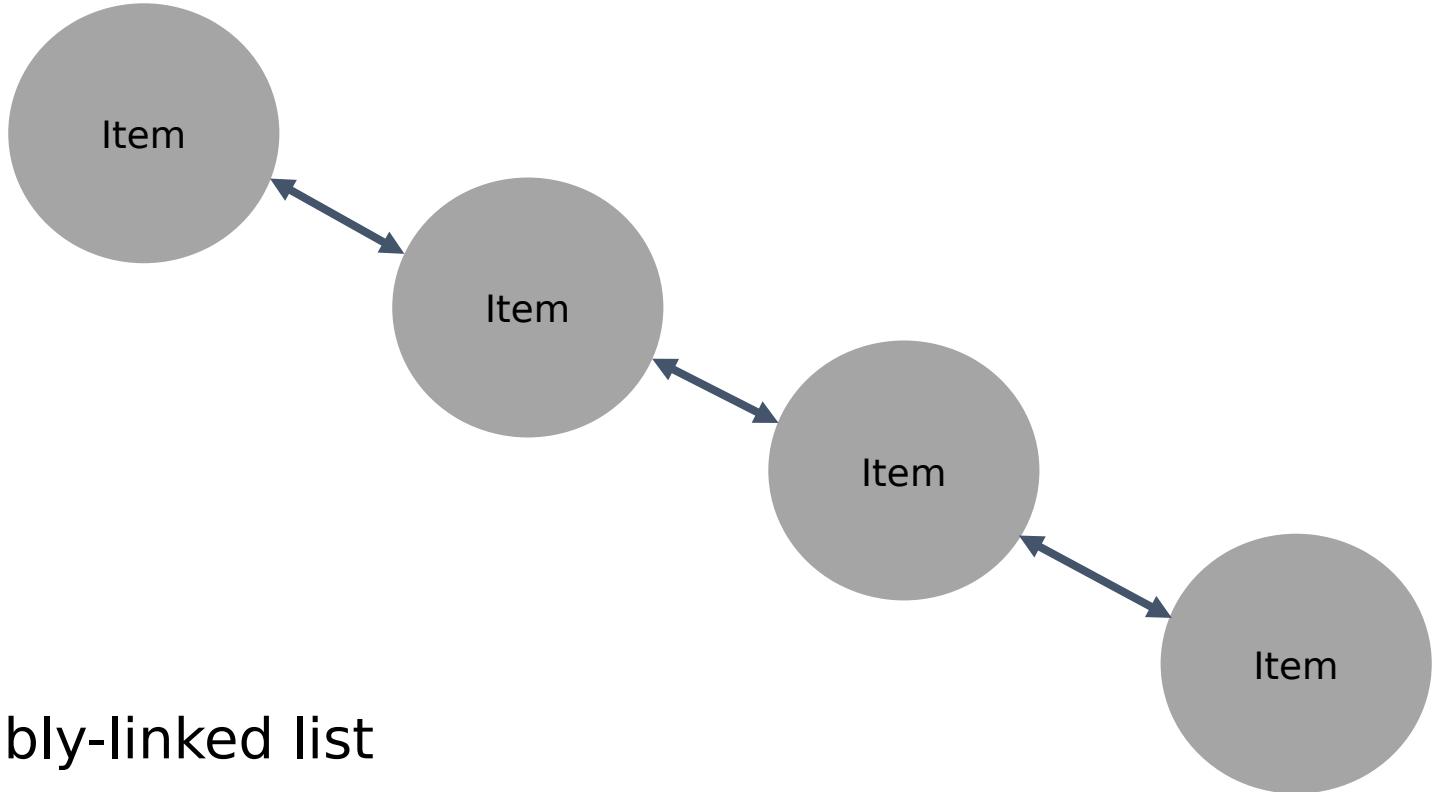




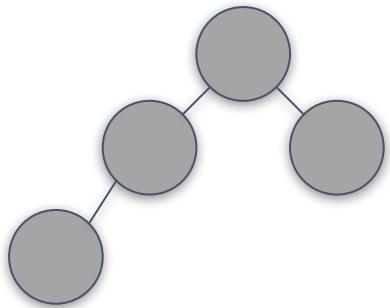
DeepLearning.AI

Leveraging LLMs for code analysis

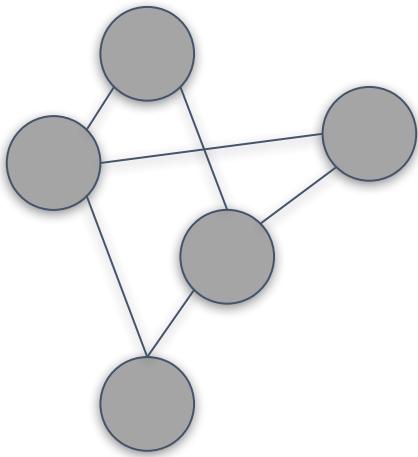
Trees



Doubly-linked list



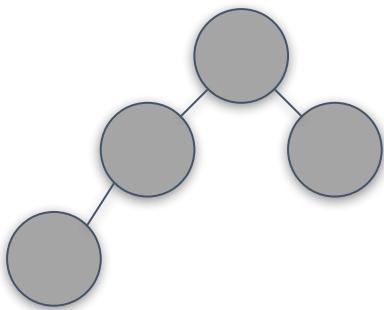
Tree



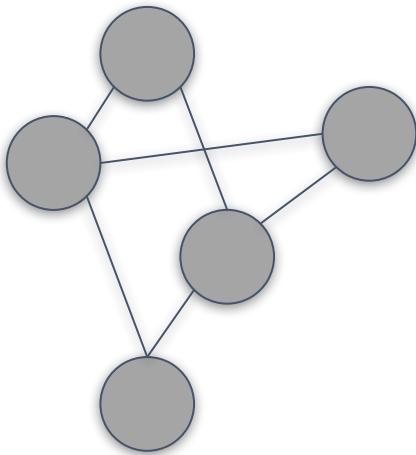
Graph

Key 1		Values
Key 2		Values
Key 3		Values
...	
Key N		Values

Hash Table



Tree

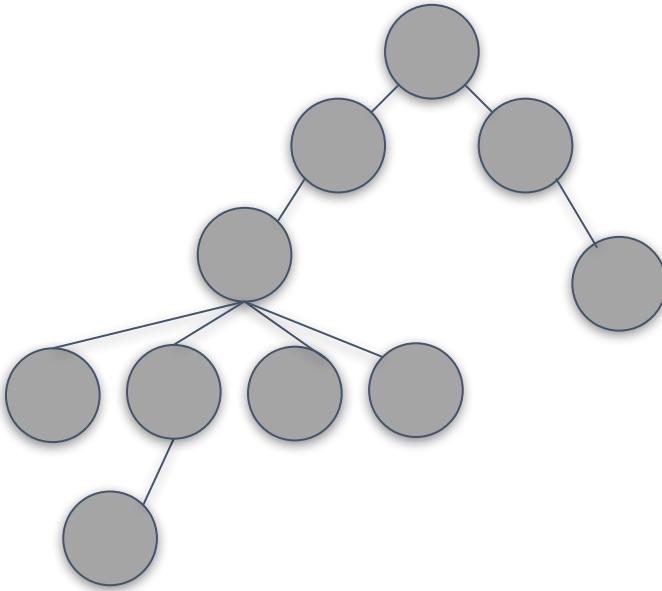


Graph

Key 1		Values
Key 2		Values
Key 3		Values
...	
Key N		Values

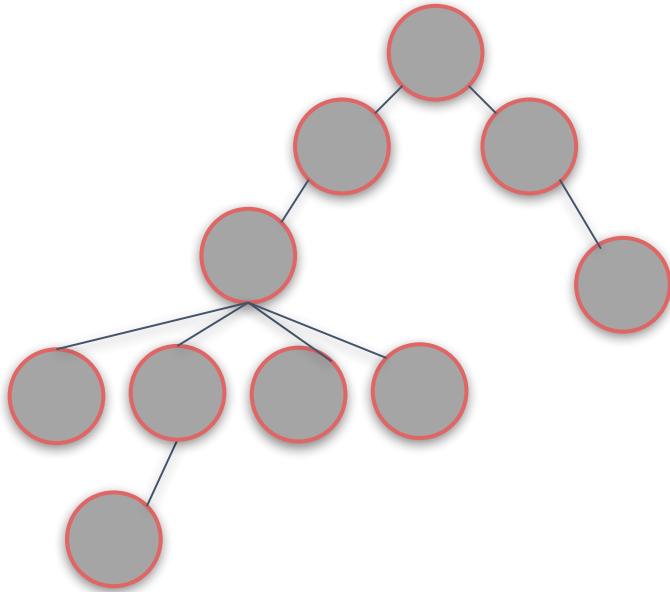
Hash Table

Trees



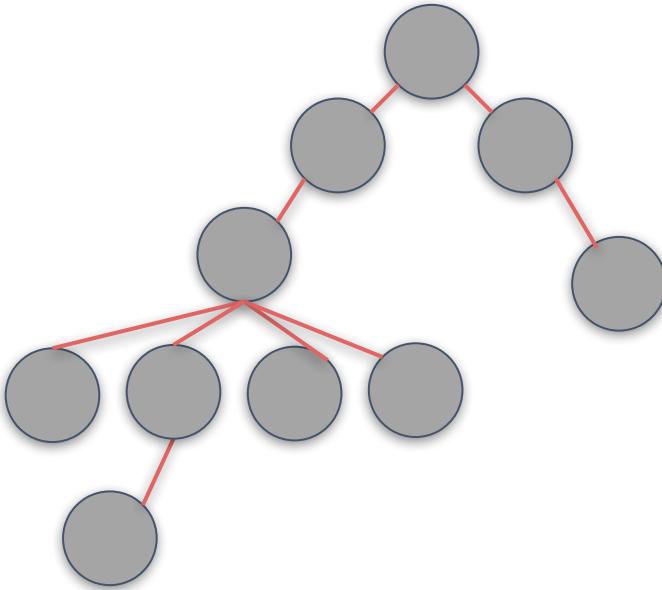
Trees

Nodes



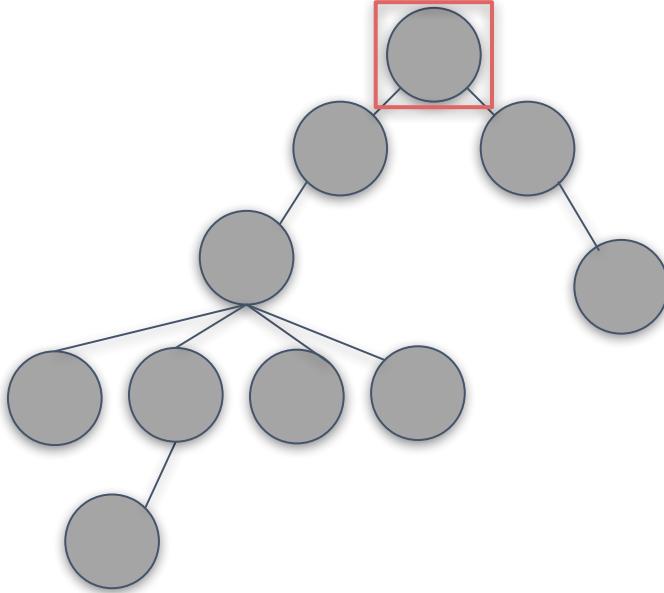
Trees

Edges



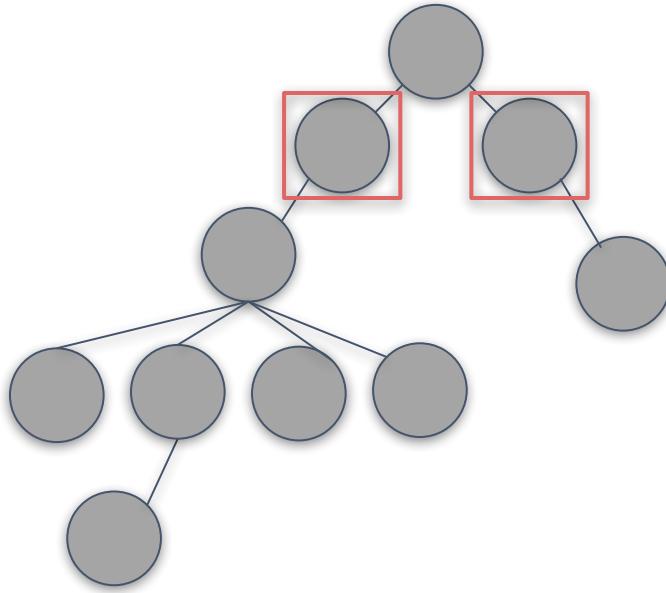
Trees

Root Node



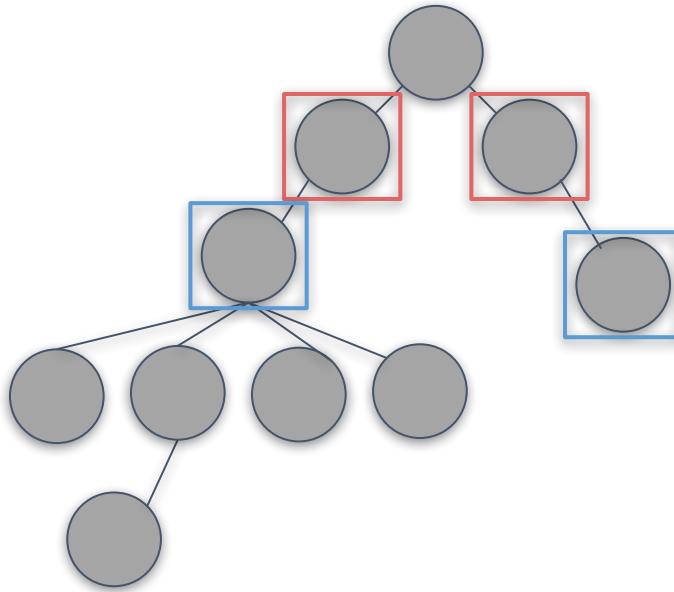
Trees

Child Nodes



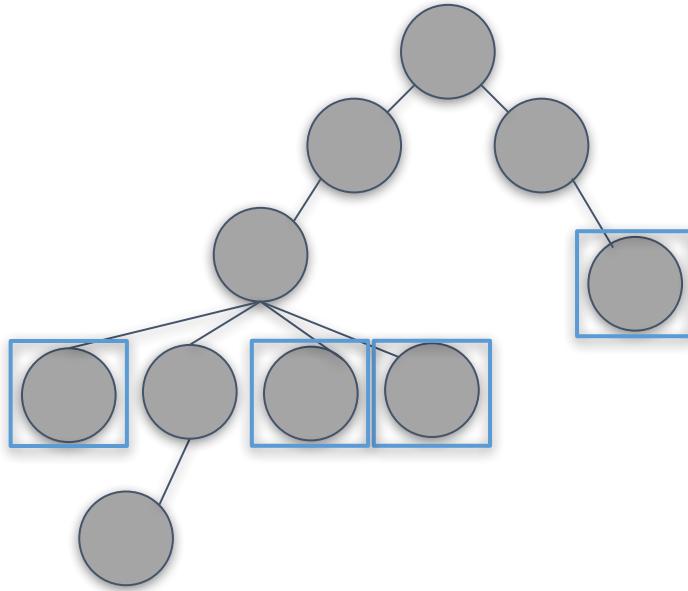
Trees

Child Nodes



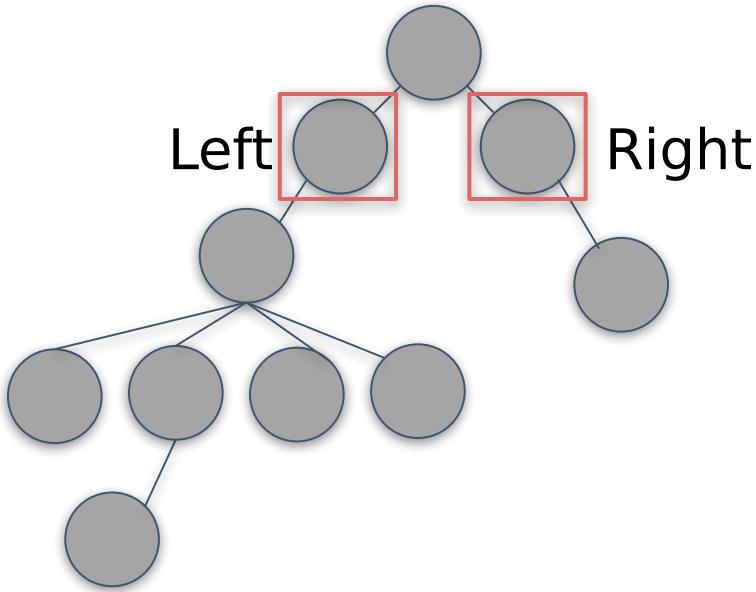
Trees

Leaf Nodes



Binary tree

Each node has at most 2 children



```
class TreeNode:  
    def __init__(self, data):  
        self.data = data  
        self.left = None  
        self.right = None
```

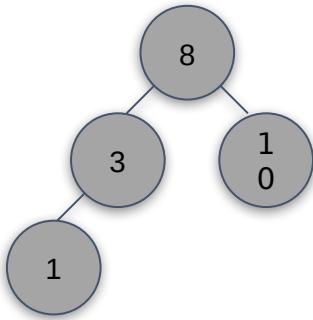
Example Usage:

```
# Creating tree nodes  
root = TreeNode("A")  
node_b = TreeNode("B")  
node_c = TreeNode("C")  
node_d = TreeNode("D")  
node_e = TreeNode("E")  
node_f = TreeNode("F")  
node_g = TreeNode("G")
```

Connecting nodes

```
root.left = node_b  
root.right = node_c  
node_b.left = node_d  
node_b.right = node_e  
node_c.left = node_f  
node_c.right = node_g
```

Binary Search Trees



```

class TreeNode:
    def __init__(self, key):
        self.left = None
        self.right = None
        self.val = key

class BinaryTree:
    def __init__(self):
        self.root = None

    def insert(self, key):
        if self.root is None:
            self.root = TreeNode(key)
        else:
            self._insert(self.root, key)

    def _insert(self, node, key):
        if key < node.val:
            if node.left is None:
                node.left = TreeNode(key)
            else:
                self._insert(node.left, key)
        else:
            if node.right is None:
                node.right = TreeNode(key)
            else:
                self._insert(node.right, key)

```

```

def inorder(self, node):
    if node:
        self.inorder(node.left)
        print(node.val, end=' ')
        self.inorder(node.right)

```

```

# Example usage
bt = BinaryTree()
bt.insert(8)
bt.insert(3)
bt.insert(10)
bt.insert(1)
bt.insert(6)
bt.insert(4)
bt.insert(7)

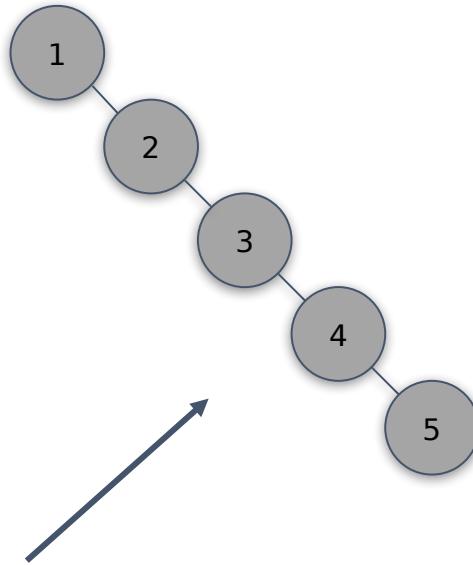
```

```

print("Inorder traversal of the binary tree:")
bt.inorder(bt.root)

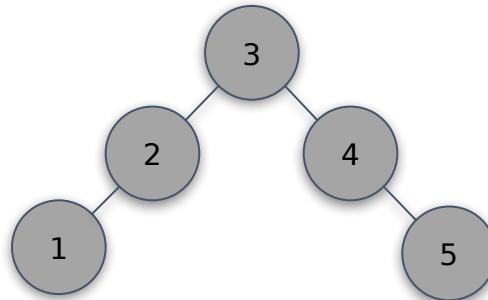
```

This is just a linked
list!



Balanced Binary Tree

Each *layer* has at most 2^n nodes, where n is the layer depth



$n = 0$

$n = 1$

$n = 2$

Analyzing the performance of trees

Prompt: You are an expert software engineer and site reliability engineer that puts code into production in large scale systems.

I have an implementation of a binary search tree here — please analyze it and let me know what I would need to do to trust this in a production environment, particularly from a security perspective.

Also, is there anything missing I should add?

Analyzing the performance of trees

Prompt: You are an expert software engineer and site reliability engineer that puts code into production in large scale systems.

I have an implementation of a binary search tree here — please analyze it and let me know what I would need to do to trust this in a production environment, particularly from a security perspective.

Also, is there anything missing I should add?

Analyzing the performance of trees

Prompts: Can you update the code to fix all of the security considerations, while also adding the functional enhancement?

Please thoroughly document this code line by line to explain what each line does. Be very detailed, in particular for the recursive and threaded parts

An algorithm for the organization of information
by G. M. Adel'son-Vel'ski and E. M. Landis
Soviet Mathematics Doklady, 3, 1259-1262, 1962
Thus (5.1) is not true for this function for $l = 1$ and $r = 1$.
<http://monet.sku.ac.kr>
Networking Lab.

Received 22/MAR/62

BIBLIOGRAPHY

- [1] C. Miranda, *Equazioni alle derivate parziali di tipo ellittico*, Springer, Berlin, 1955.
- [2] S. M. Nikol'ski, Sibirsk. Mat. Z. 1 (1960), 78.

Translated by:
F. M. Goodspeed

AN ALGORITHM FOR THE ORGANIZATION OF INFORMATION

G. M. ADEL'SON-VEL'SKI AND E. M. LANDIS

In the present article we discuss the organization of information contained in the cells of an automatic calculating machine. A three-address machine will be used for this study.

Statement of the problem. The information enters a machine in sequence from a certain reserve. The information element is contained in a group of cells which are arranged one after the other. A certain number (the information estimate), which is different for different elements, is contained in the information element. The information must be organized in the memory of the machine in such a way that at any moment a very large number of operations is not required to scan the information with the given evaluation and to record the new information element.

An algorithm is proposed in which both the search and the recording are carried out in $C \lg N$ operations, where N is the number of information elements which have entered at a given moment.

A part of the memory of the machine is set aside to store the incoming information. The information elements are arranged there in their order of entry. Moreover, in another part of the memory a "reference board" [1] is formed, each cell of which corresponds to one of the information elements.

The reference board is a dyadic tree (Figure 1a): each of its cells has no more than one left cell, and no more than one right cell subordinate to it. Direct subordination induces subordination (partial ordering). In addition, for each cell of the tree, all the cells which are subordinate to a left (right) directly subordinate cell, will be arranged further to the left (right) than the given cell. Moreover, we assume that there is a cell (the head) to which all the others are subordinate. By transitivity, the conception "further to the left" and "further to the right" extends to the aggregate of all the cell pairs, and this aggregate becomes ordered. Thus, a given order of cells in a reference board should coincide with the order of arrangement of the estimates of the corresponding information elements (to be specific, we shall consider the estimates as increasing from left to right).

In the first address of each cell of the reference board, a place is indicated where the corresponding information element is located. The addresses of the cells of the reference board, which are directly subordinate on the left and right respectively to the given cell, are located in the second and third addresses. If a cell has no directly subordinate cells on either side, then there is zero in the corre-

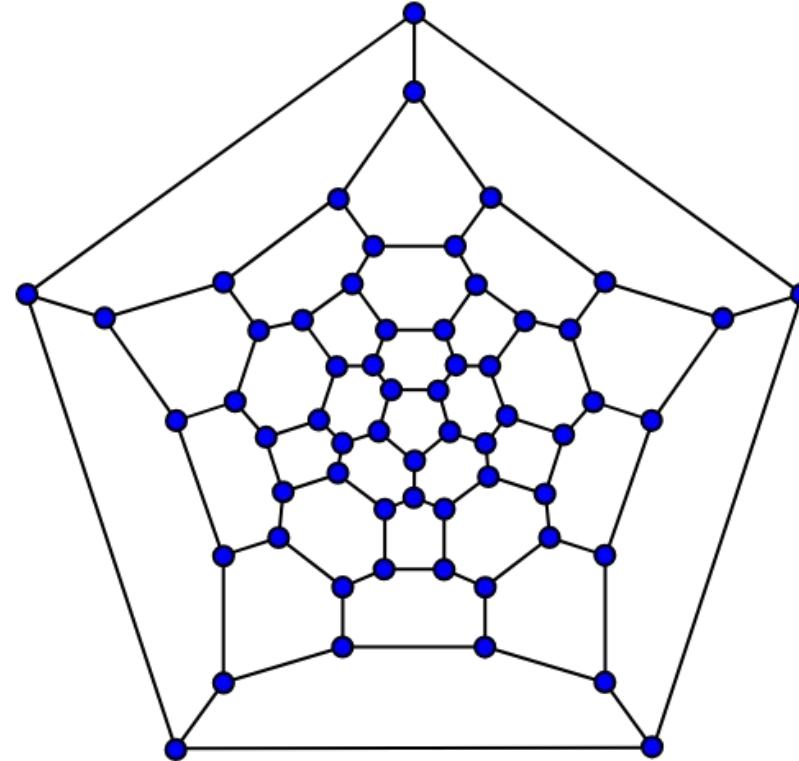


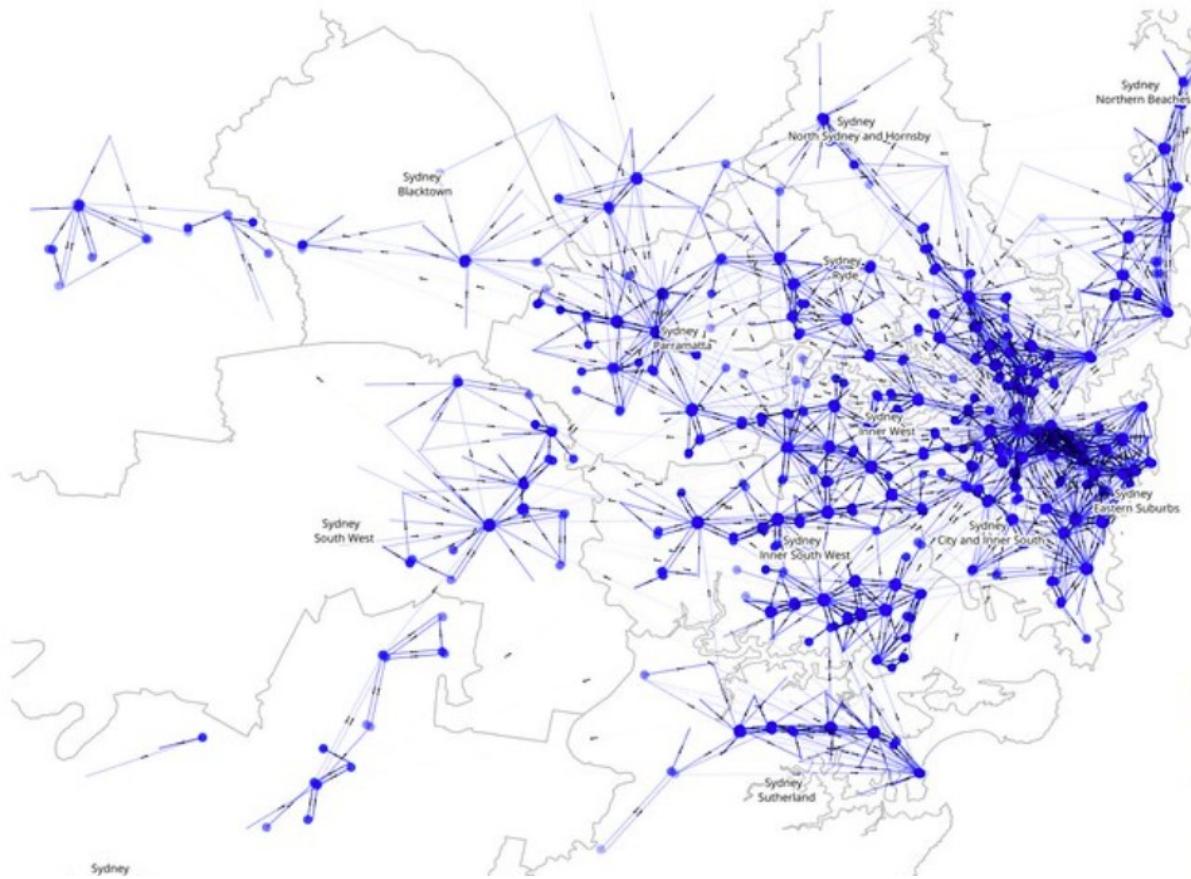
DeepLearning.AI

Leveraging LLMs for code analysis

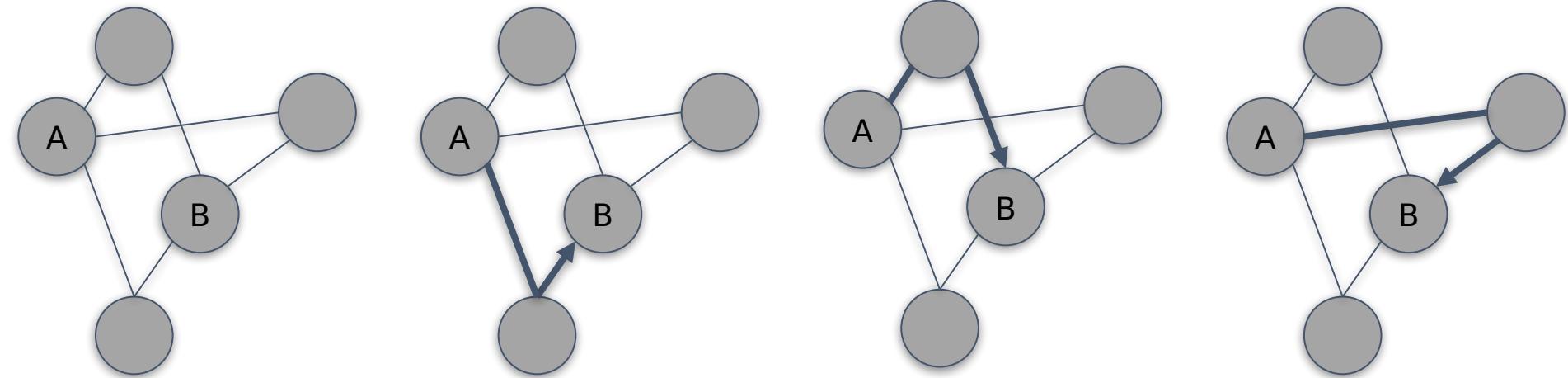
Graphs

Graphs

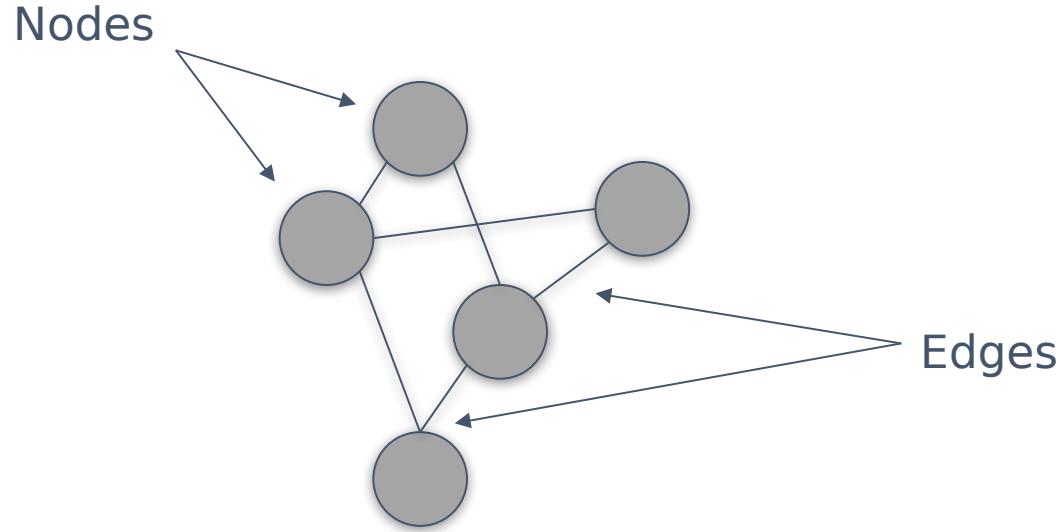




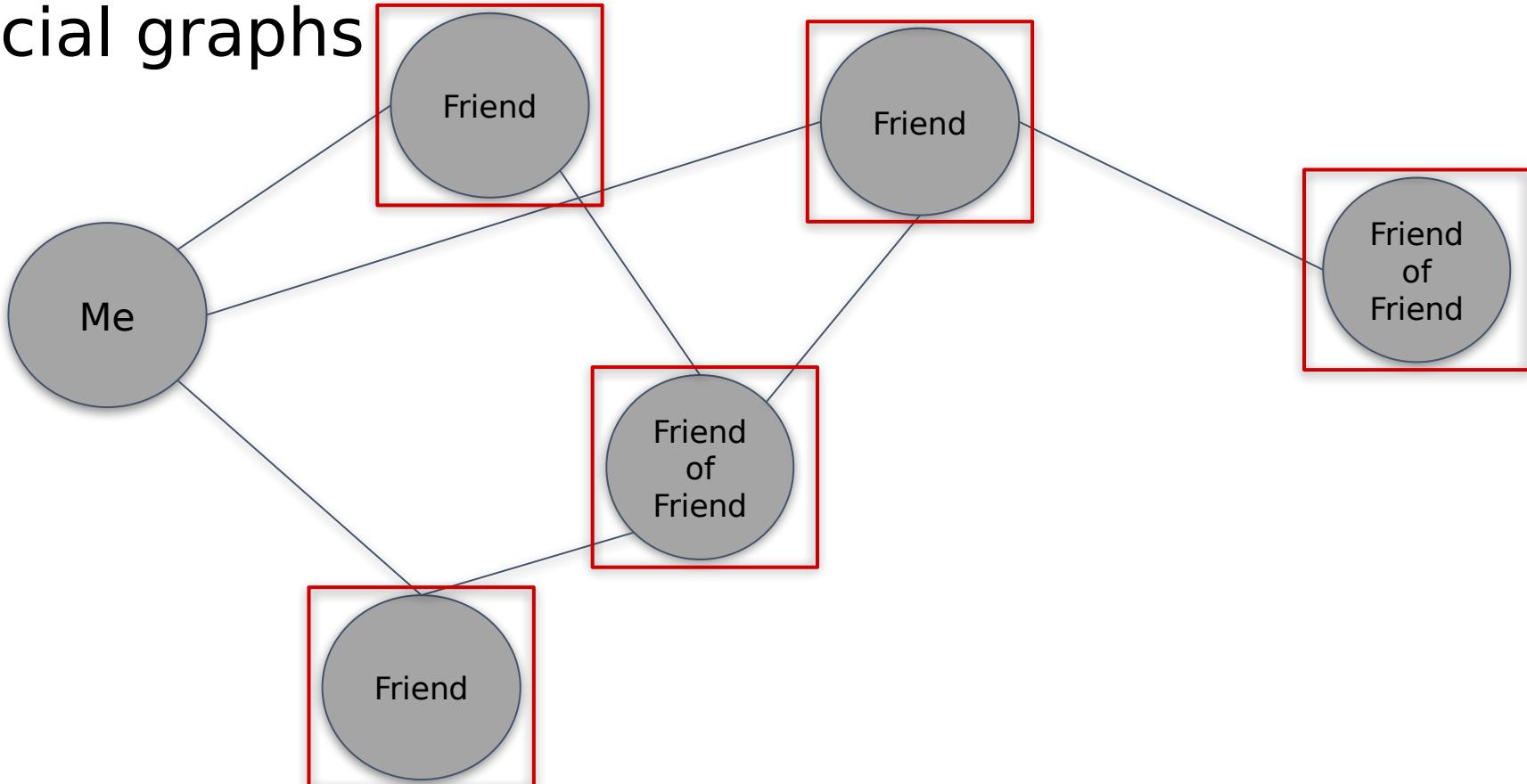
Graphs



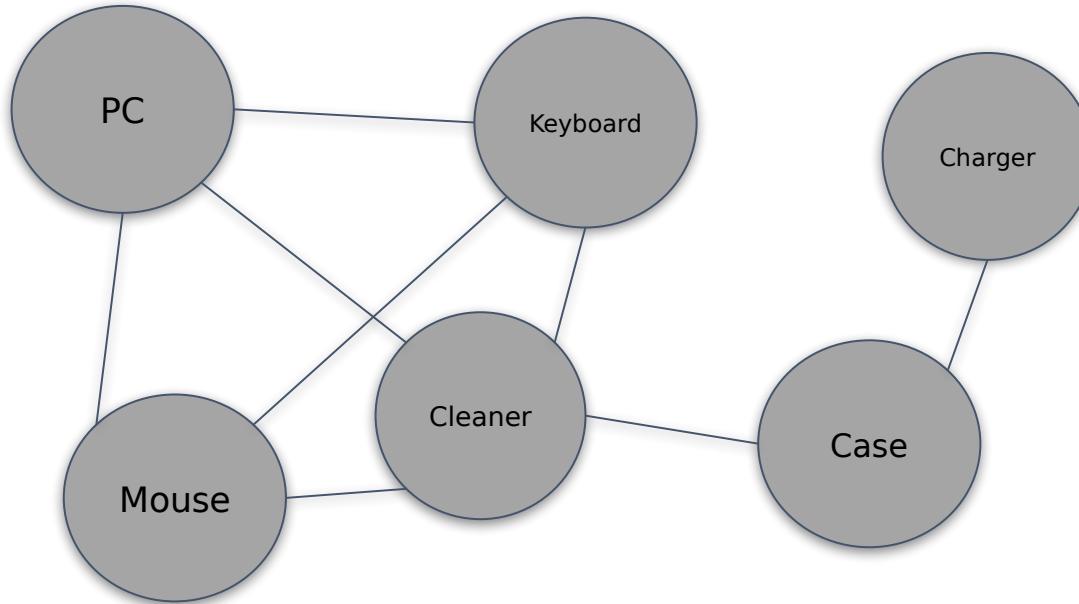
Graphs



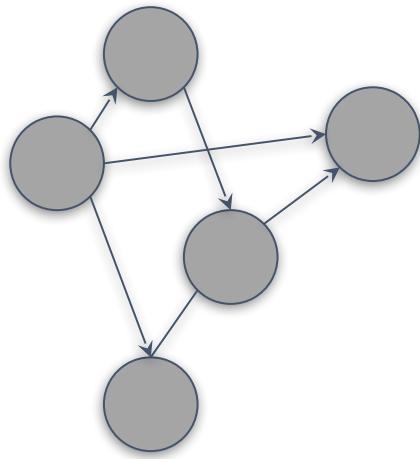
Social graphs



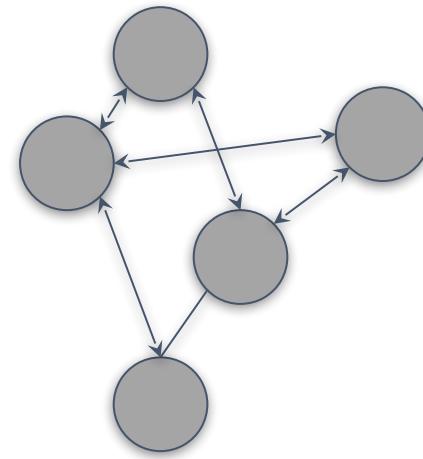
Shopping graphs



Directed vs. undirected graphs

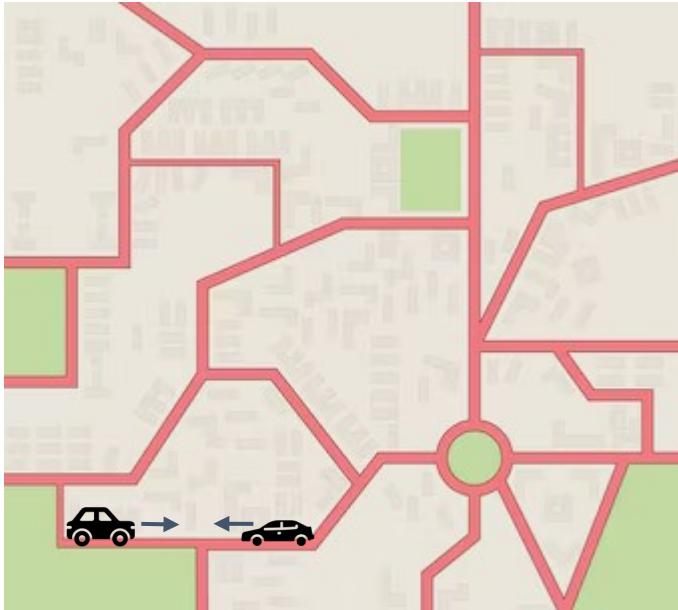


Directed Graph



Undirected Graph

Directed vs. undirected graphs



```

class Graph:
    def __init__(self, directed=False):
        self.graph = {}
        self.directed = directed

    def add_vertex(self, vertex):
        if vertex not in self.graph:
            self.graph[vertex] = []

    def add_edge(self, src, dest):
        if src not in self.graph:
            self.add_vertex(src)
        if dest not in self.graph:
            self.add_vertex(dest)
        self.graph[src].append(dest)
        if not self.directed:
            self.graph[dest].append(src)

    def remove_edge(self, src, dest):
        if src in self.graph:
            if dest in self.graph[src]:
                self.graph[src].remove(dest)
        if not self.directed:
            if dest in self.graph and src in self.graph[dest]:
                self.graph[dest].remove(src)

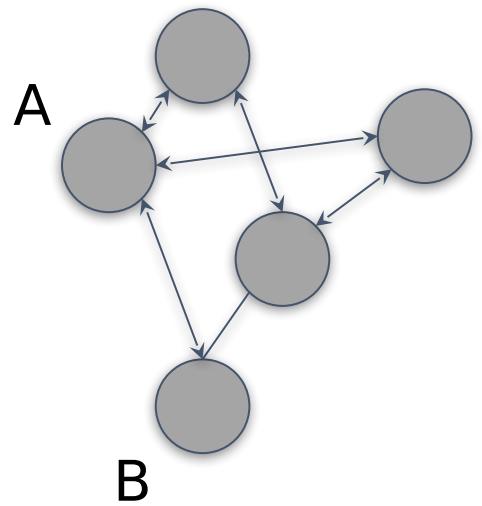
    def remove_vertex(self, vertex):
        if vertex in self.graph:
            # Remove any edges from other vertices to this one
            for adj in list(self.graph):
                if vertex in self.graph[adj]:
                    self.graph[adj].remove(vertex)
            # Remove the vertex entry
            del self.graph[vertex]

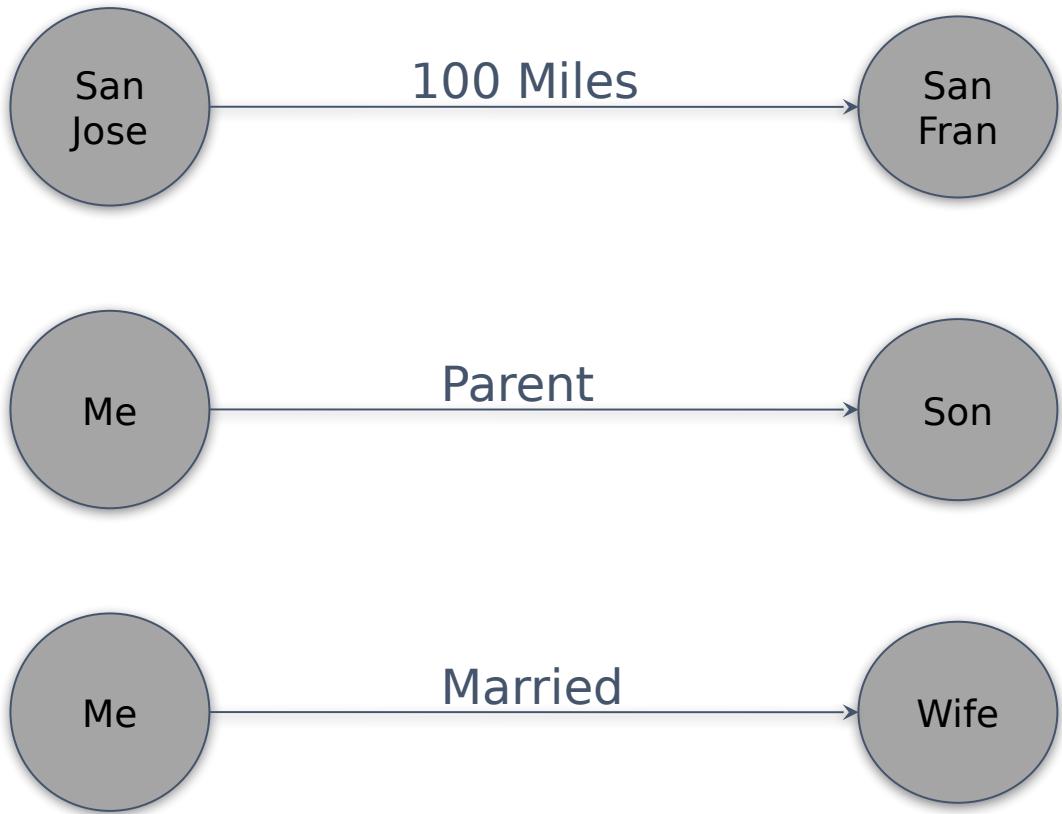
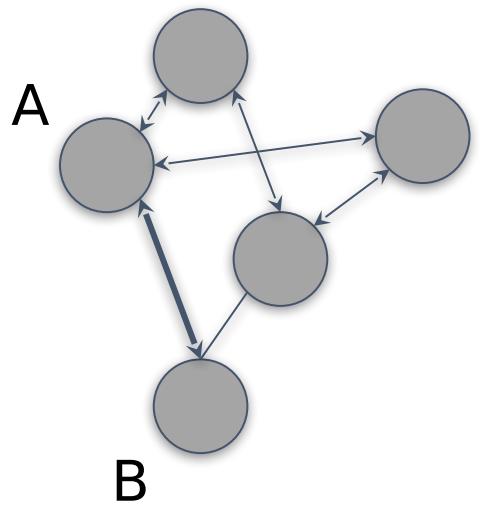
    def get_adjacent_vertices(self, vertex):
        if vertex in self.graph:
            return self.graph[vertex]
        else:
            return []

    def __str__(self):
        return str(self.graph)

# Example usage:
g = Graph(directed=True)
g.add_vertex('A')
g.add_vertex('B')
g.add_edge('A', 'B')
g.add_edge('A', 'C')
print(g)

```





```

class Graph:
    def __init__(self, directed=False):
        self.graph = {}
        self.directed = directed

    def add_vertex(self, vertex):
        if vertex not in self.graph:
            self.graph[vertex] = []

    def add_edge(self, src, dest):
        if src not in self.graph:
            self.add_vertex(src)
        if dest not in self.graph:
            self.add_vertex(dest)
        self.graph[src].append(dest)
        if not self.directed:
            self.graph[dest].append(src)

    def remove_edge(self, src, dest):
        if src in self.graph:
            if dest in self.graph[src]:
                self.graph[src].remove(dest)
        if not self.directed:
            if dest in self.graph and src in self.graph[dest]:
                self.graph[dest].remove(src)

    def remove_vertex(self, vertex):
        if vertex in self.graph:
            # Remove any edges from other vertices to this one
            for adj in list(self.graph):
                if vertex in self.graph[adj]:
                    self.graph[adj].remove(vertex)
            # Remove the vertex entry
            del self.graph[vertex]

    def get_adjacent_vertices(self, vertex):
        if vertex in self.graph:
            return self.graph[vertex]
        else:
            return []

    def __str__(self):
        return str(self.graph)

# Example usage:
g = Graph(directed=True)
g.add_vertex('A')
g.add_vertex('B')
g.add_edge('A', 'B')
g.add_edge('A', 'C')
print(g)

```

Analyzing the performance of graphs

Prompt: You are an expert software developer or site reliability engineer, and your job is to ensure that this code runs effectively, quickly, at scale, and securely. Please profile it, and find any issues that need to be fixed or updated.

Analyzing the performance of graphs

Prompt: Please go through and edit the code to implement the recommendations, add input validation, and make it as memory efficient as possible.

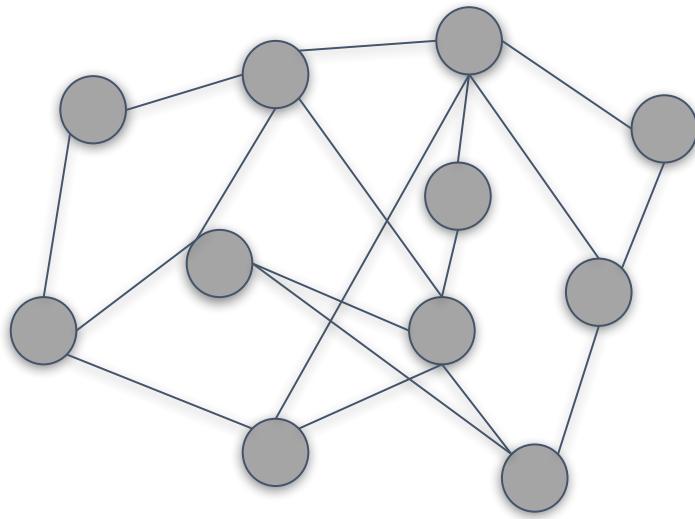


DeepLearning.AI

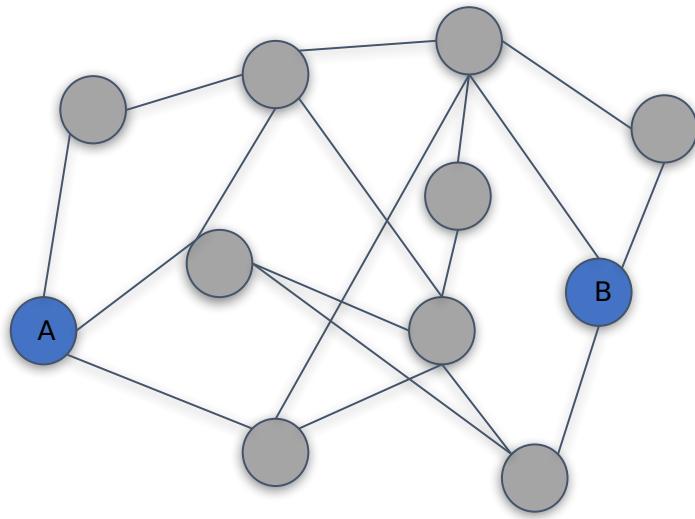
Leveraging LLMs for code analysis

Graded lab:
Graph algorithms

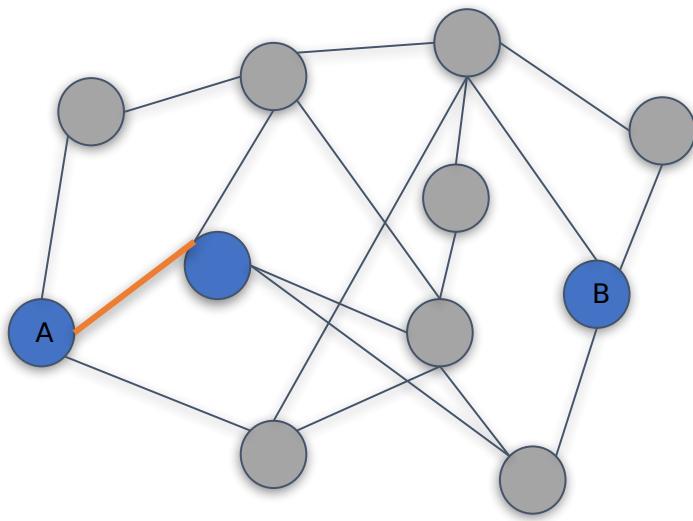
Problem 1: Shortest route between nodes



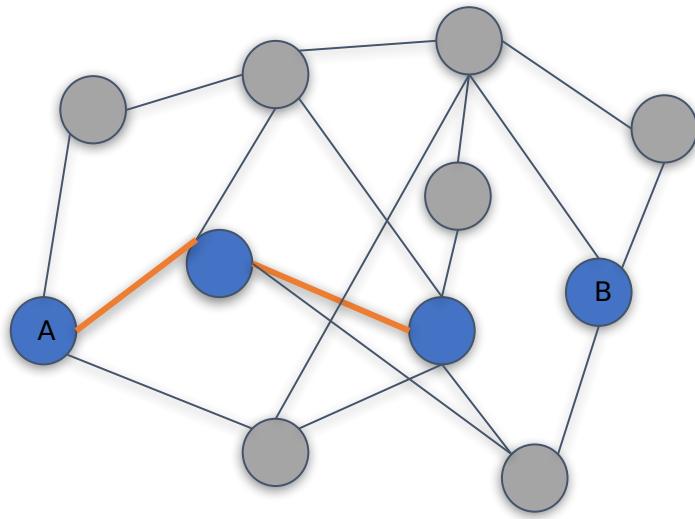
Problem 1: Shortest route between nodes



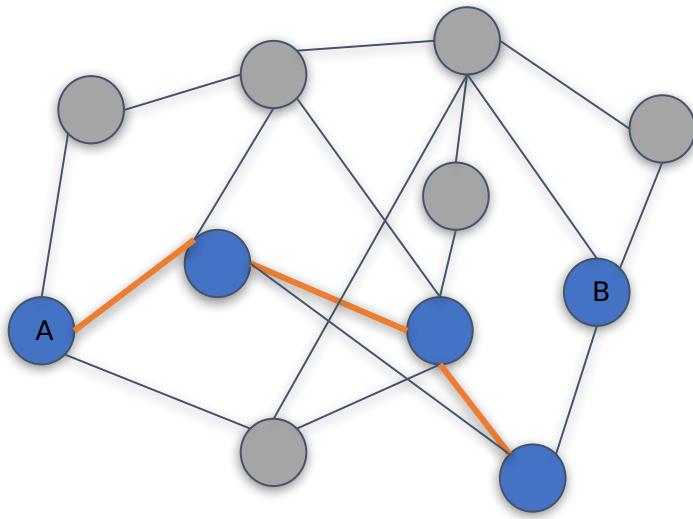
Problem 1: Shortest route between nodes



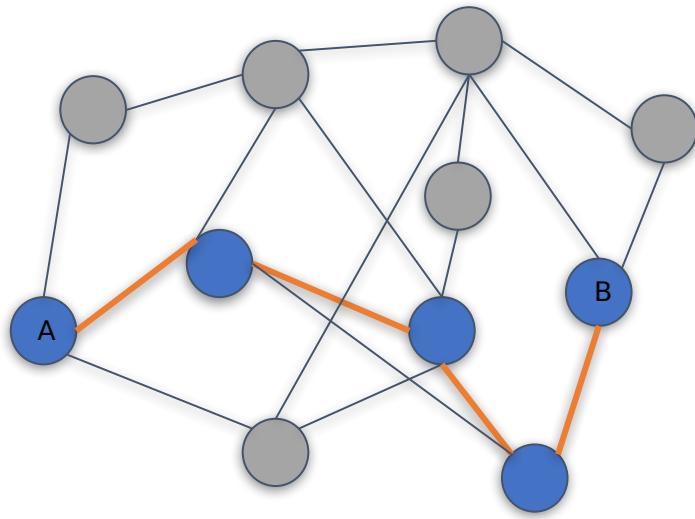
Problem 1: Shortest route between nodes



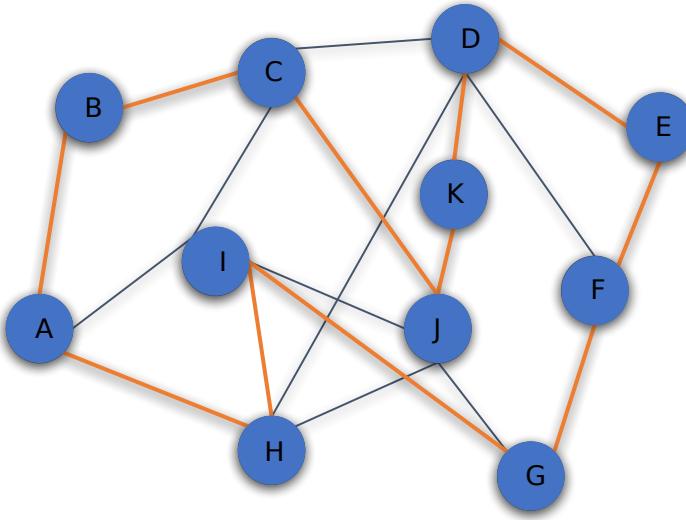
Problem 1: Shortest route between nodes



Problem 1: Shortest route between nodes



Problem 2: Traveling Salesman





DeepLearning.AI

Leveraging LLMs for code analysis

Hash tables and maps

player_homeruns[0]=17

player_homeruns[1]=1

player_homeruns[2]=4

```
player_homeruns["Bob"] = 17
```

```
player_homeruns["Jim"] = 1
```

```
player_homeruns["Dave"] = 4
```

Thunder and Lightning. Enter three Witches.

FIRST WITCH

When shall we three meet again?
In thunder, lightning, or in rain?

SECOND WITCH

When the hurly-burly's done,
When the battle's lost and won.

THIRD WITCH

That will be ere the set of sun.

Thunder and Lightning. Enter three Witches.

FIRST WITCH

When shall we three meet again?
In thunder, lightning, or in rain?

```
word_count["thunder"]++
```

SECOND WITCH

When the hurly-burly's done,
When the battle's lost and won.

THIRD WITCH

That will be ere the set of sun.



Thunder and *Lightning*. Enter three Witches.

FIRST WITCH

When shall we three meet again?
In thunder, lightning, or in rain?

```
word_count["thunder"]++
```

SECOND WITCH

When the hurly-burly's done,
When the battle's lost and won.

THIRD WITCH

That will be ere the set of sun.



Thunder and Lightning. Enter three Witches.

FIRST WITCH

When shall we three meet again?
In thunder, lightning, or in rain?

SECOND WITCH

When the hurly-burly's done,
When the battle's lost and won.

THIRD WITCH

That will be ere the set of sun.

```
for each word in corpus:  
if word_count[word]:  
    word_count[word]++  
else:  
    word_count[word] = 1
```



Counting words in text

Prompt: Please write plain Python code using the base language to download text from a given URL, and then count every instance of every word in that text.

```
from collections import Counter  
words = re.findall(r'\b\w+\b', text.lower())  
word_count = Counter(words)
```

```
from collections import Counter  
words = re.findall(r'\b\w+\b', text.lower())  
word_count = Counter(words)
```

Asking clarifying questions

Prompt: I'm not familiar with counter in Python. Please explain what it is.

```
from collections import Counter  
words = re.findall(r'\b\w+\b', text.lower())  
word_count = Counter(words)
```

Asking clarifying questions

Prompt: Why did you use a counter to do this, instead of just a Python dictionary?

Asking clarifying questions

Prompt: Say I had millions of URLs that I needed to count the words in. Does counter scale better than dictionary for that?

Asking clarifying questions

Prompt: I see you use a regular expression to break the corpus into extra words. Is this the fastest and most scalable way? What if there were millions of books instead of just one?

Using a Counter from the collections module provides several advantages over a standard Python dictionary for counting occurrences of items. Here are some reasons why Counter is a better choice in this context:

Readability and Conciseness:

Counter is specifically designed for counting and makes the code more readable and concise. Using Counter, you can directly count elements with a single line, whereas using a dictionary would require more lines of code for initialization and updating counts.

Built-in Methods:

Counter provides several convenient methods for common tasks:

- `most_common()`: returns a list of the most common elements and their counts.
- `elements()`: returns an iterator over elements, repeating each as many times as its count.

Arithmetic operations: Counter supports addition, subtraction, intersection, and union operations, which can be useful for more advanced counting tasks.

Default Value Handling:

With a Counter, you don't need to check if a key is already present. It handles this automatically, which simplifies the code. In contrast, using a dictionary would require explicit checks or the use of `defaultdict`.

```
import urllib.request
from collections import Counter
import re
from multiprocessing import Pool

def download_and_count(url):
    try:
        response = urllib.request.urlopen(url)
        text = response.read().decode('utf-8')
        words = re.findall(r'\b\w+\b', text.lower())
        return Counter(words)
    except Exception as e:
        print(f"Error processing {url}: {e}")
        return Counter()

def merge_counters(counters):
    total_count = Counter()
    for counter in counters:
        total_count.update(counter)
    return total_count

def main(urls):
    with Pool(processes=8) as pool: # Adjust the number of processes as needed
        counters = pool.map(download_and_count, urls)
    total_word_count = merge_counters(counters)
    for word, count in total_word_count.items():
        print(f'{word}: {count}'
```

1. Unverified URL Inputs:

The URLs are directly used without validation or sanitization, which could lead to issues like SSRF (Server-Side Request Forgery).

2. Error Handling:

The exception handling is too broad. Catching all exceptions and printing them can hide specific issues that should be addressed separately.

Returning an empty Counter on failure might lead to bad results w/o proper notification.

3. Concurrency Issues:

Using multiprocessing with external resources (like network calls) can lead to potential race conditions or overwhelming the target servers.

4. Resource Management:

There is no timeout for `urllib.request.urlopen`, which might cause the script to hang indefinitely if a server doesn't respond.

5. Lack of Logging:

There is no proper logging mechanism. Using print statements is not suitable for debugging or monitoring in a production environment.

6. Regular Expression Use:

- The regex `r'\b\w+\b'` might not be robust enough to handle all kinds of text inputs (e.g., it might not correctly handle words with apostrophes or other punctuation).

1. Unverified URL Inputs:

The URLs are directly used without validation or sanitization, which could lead to issues like SSRF (Server-Side Request Forgery).

2. Error Handling:

The exception handling is too broad. Catching all exceptions and printing them can hide specific issues that should be addressed separately.

Returning an empty Counter on failure might lead to bad results w/o proper notification.

3. Concurrency Issues:

Using multiprocessing with external resources (like network calls) can lead to potential race conditions or overwhelming the target servers.

4. Resource Management:

There is no timeout for `urllib.request.urlopen`, which might cause the script to hang indefinitely if a server doesn't respond.

5. Lack of Logging:

There is no proper logging mechanism. Using print statements is not suitable for debugging or monitoring in a production environment.

6. Regular Expression Use:

- The regex `r'\b\w+\b'` might not be robust enough to handle all kinds of text inputs (e.g., it might not correctly handle words with apostrophes or other punctuation).

1. Unverified URL Inputs:

The URLs are directly used without validation or sanitization, which could lead to issues like SSRF (Server-Side Request Forgery).

2. Error Handling:

The exception handling is too broad. Catching all exceptions and printing them can hide specific issues that should be addressed separately.

Returning an empty Counter on failure might lead to bad results w/o proper notification.

3. Concurrency Issues:

Using multiprocessing with external resources (like network calls) can lead to potential race conditions or overwhelming the target servers.

4. Resource Management:

There is no timeout for `urllib.request.urlopen`, which might cause the script to hang indefinitely if a server doesn't respond.

5. Lack of Logging:

There is no proper logging mechanism. Using print statements is not suitable for debugging or monitoring in a production environment.

6. Regular Expression Use:

- The regex `r'\b\w+\b'` might not be robust enough to handle all kinds of text inputs (e.g., it might not correctly handle words with apostrophes or other punctuation).

1. Unverified URL Inputs:

The URLs are directly used without validation or sanitization, which could lead to issues like SSRF (Server-Side Request Forgery).

2. Error Handling:

The exception handling is too broad. Catching all exceptions and printing them can hide specific issues that should be addressed separately.

Returning an empty Counter on failure might lead to bad results w/o proper notification.

3. Concurrency Issues:

Using multiprocessing with external resources (like network calls) can lead to potential race conditions or overwhelming the target servers.

4. Resource Management:

There is no timeout for `urllib.request.urlopen`, which might cause the script to hang indefinitely if a server doesn't respond.

5. Lack of Logging:

There is no proper logging mechanism. Using print statements is not suitable for debugging or monitoring in a production environment.

6. Regular Expression Use:

- The regex `r'\b\w+\b'` might not be robust enough to handle all kinds of text inputs (e.g., it might not correctly handle words with apostrophes or other punctuation).

1. Unverified URL Inputs:

The URLs are directly used without validation or sanitization, which could lead to issues like SSRF (Server-Side Request Forgery).

2. Error Handling:

The exception handling is too broad. Catching all exceptions and printing them can hide specific issues that should be addressed separately.

Returning an empty Counter on failure might lead to bad results w/o proper notification.

3. Concurrency Issues:

Using multiprocessing with external resources (like network calls) can lead to potential race conditions or overwhelming the target servers.

4. Resource Management:

There is no timeout for `urllib.request.urlopen`, which might cause the script to hang indefinitely if a server doesn't respond.

5. Lack of Logging:

There is no proper logging mechanism. Using print statements is not suitable for debugging or monitoring in a production environment.

6. Regular Expression Use:

- The regex `r'\b\w+\b'` might not be robust enough to handle all kinds of text inputs (e.g., it might not correctly handle words with apostrophes or other punctuation).

1. Unverified URL Inputs:

The URLs are directly used without validation or sanitization, which could lead to issues like SSRF (Server-Side Request Forgery).

2. Error Handling:

The exception handling is too broad. Catching all exceptions and printing them can hide specific issues that should be addressed separately.

Returning an empty Counter on failure might lead to bad results w/o proper notification.

3. Concurrency Issues:

Using multiprocessing with external resources (like network calls) can lead to potential race conditions or overwhelming the target servers.

4. Resource Management:

There is no timeout for `urllib.request.urlopen`, which might cause the script to hang indefinitely if a server doesn't respond.

5. Lack of Logging:

There is no proper logging mechanism. Using print statements is not suitable for debugging or monitoring in a production environment.

6. Regular Expression Use:

- The regex `r'\b\w+\b'` might not be robust enough to handle all kinds of text inputs (e.g., it might not correctly handle words with apostrophes or other punctuation).

Improvements to explore

- Test Cases
 - Lots of different URLs and URL types
- Non-English Regex
- Document the code
- Refactor the code
 - Other programming languages
 - Count for languages other than English
 - Japanese?



DeepLearning.AI

Leveraging LLMs for code analysis

Course conclusion



Dependency Management

Course conclusion