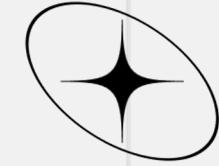


MUSA/CPLN CAPSTONE

# MULTIMODAL OPTIMIZATION OF CO<sub>2</sub> TRANSPORT IN TEXAS





# TABLE OF CONTENT

1	INTRODUCTION & BACKGROUND	5	PYTHON SCRIPT TOOLS DEVELOPED FOR RISK MODELING
2	METHODOLOGY	6	IMPROVED COST MODELING
3	STUDY AREA	8	NETWORK ANALYSIS & ROUTE OPTIMIZATION
4	RISK DATA & MODELING	9	RESULTS

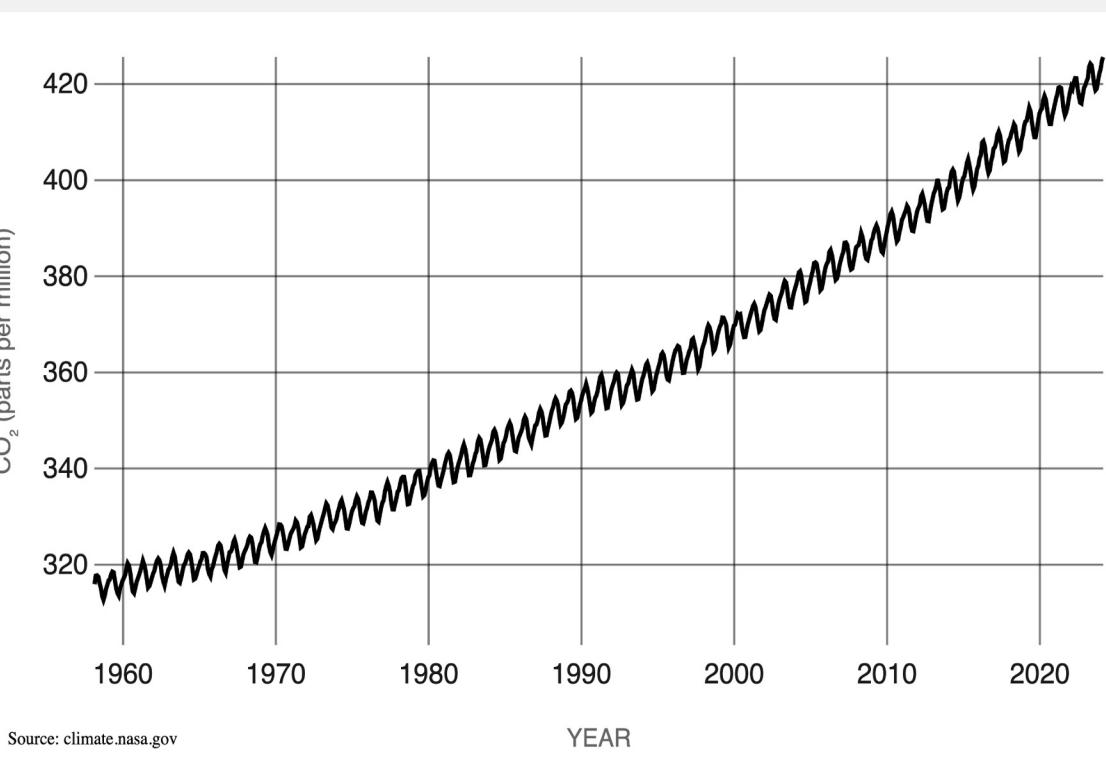


# INTRODUCTIONS

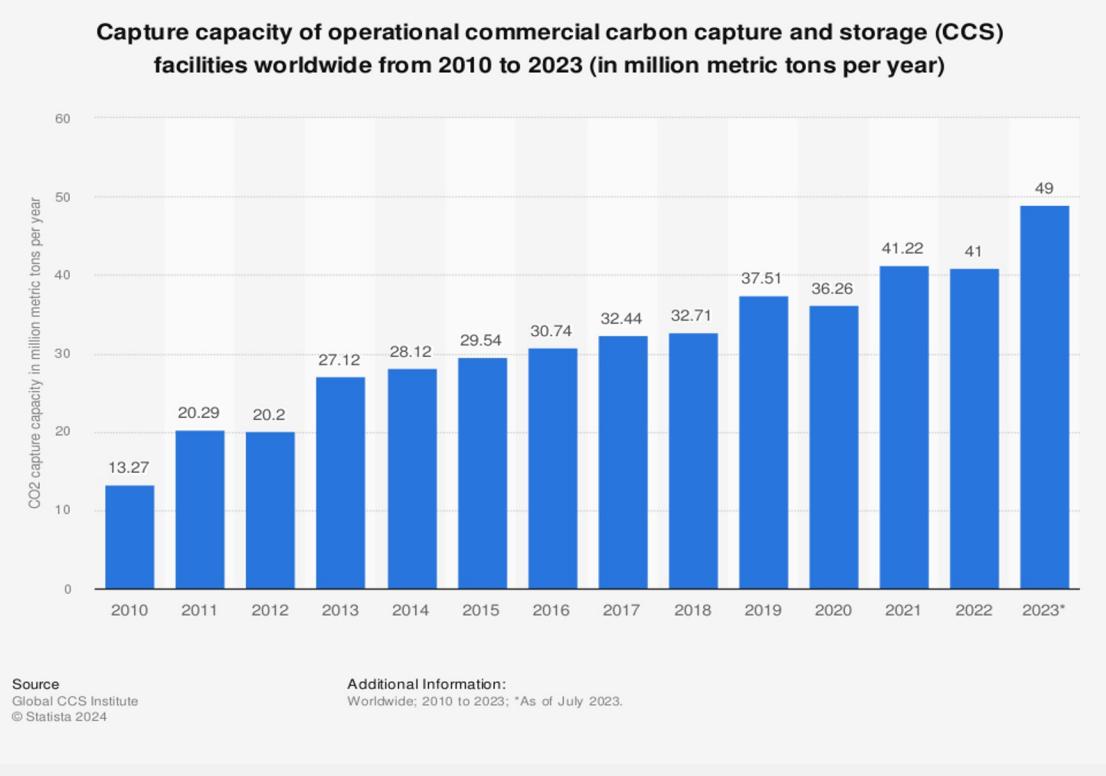
US CCS Facilities. ([Congressional Budget Office, 2023](#)).

Name of Facility	Date CCS Operations Began	Location	Type of Production	CO <sub>2</sub> Used for Enhanced Oil Recovery?	CO <sub>2</sub> Capture Capacity (Millions of metric tons per year)
Terrell	1972	Texas	Natural Gas Processing	Yes	0.5
Enid Fertilizer	1982	Oklahoma	Ammonia (Fertilizer)	Yes	0.2
Shute Creek	1986	Wyoming	Natural Gas Processing	Yes	7.0
Great Plains	2000	North Dakota	Hydrogen and Ammonia (Fertilizer) <sup>a</sup>	Yes	3.0
Core Energy	2003	Michigan	Natural Gas Processing	Yes	0.4
Arkalon	2009	Kansas	Ethanol	Yes	0.5
Century Plant	2010	Texas	Natural Gas Processing	Yes	5.0
Bonanza BioEnergy	2012	Kansas	Ethanol	Yes	0.1
Air Products	2013	Texas	Hydrogen	Yes	0.9
Coffeyville	2013	Kansas	Hydrogen and Ammonia (Fertilizer) <sup>a</sup>	Yes	0.9
Lost Cabin	2013	Wyoming	Natural Gas Processing	Yes	0.9
PCS Nitrogen	2013	Louisiana	Ammonia (Fertilizer)	Yes	0.3
Petra Nova	2017 <sup>b</sup>	Texas	Electric Power	Yes	1.4
Illinois Industrial	2017	Illinois	Ethanol	No	1.0
Red Trail Energy	2022	North Dakota	Ethanol	No	0.2

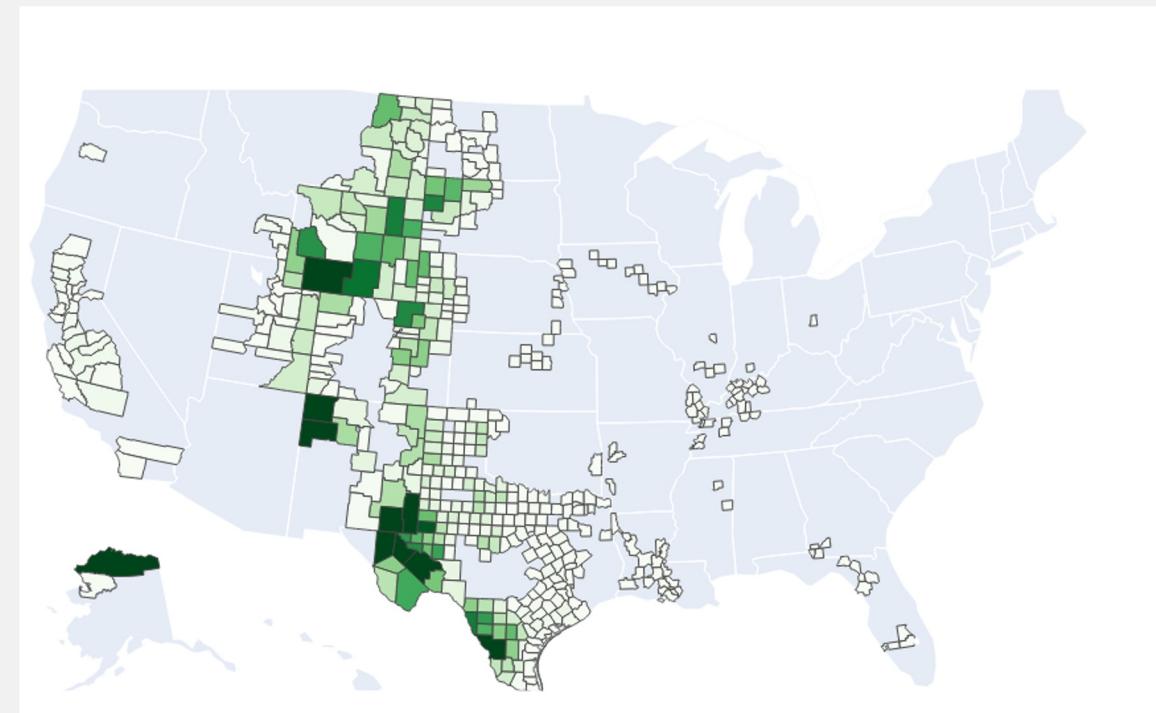
United States ranks the 1st regarding the capacity of operational carbon capture and storage facilities worldwide as of 2022, with 23.7 million metric tons per year.



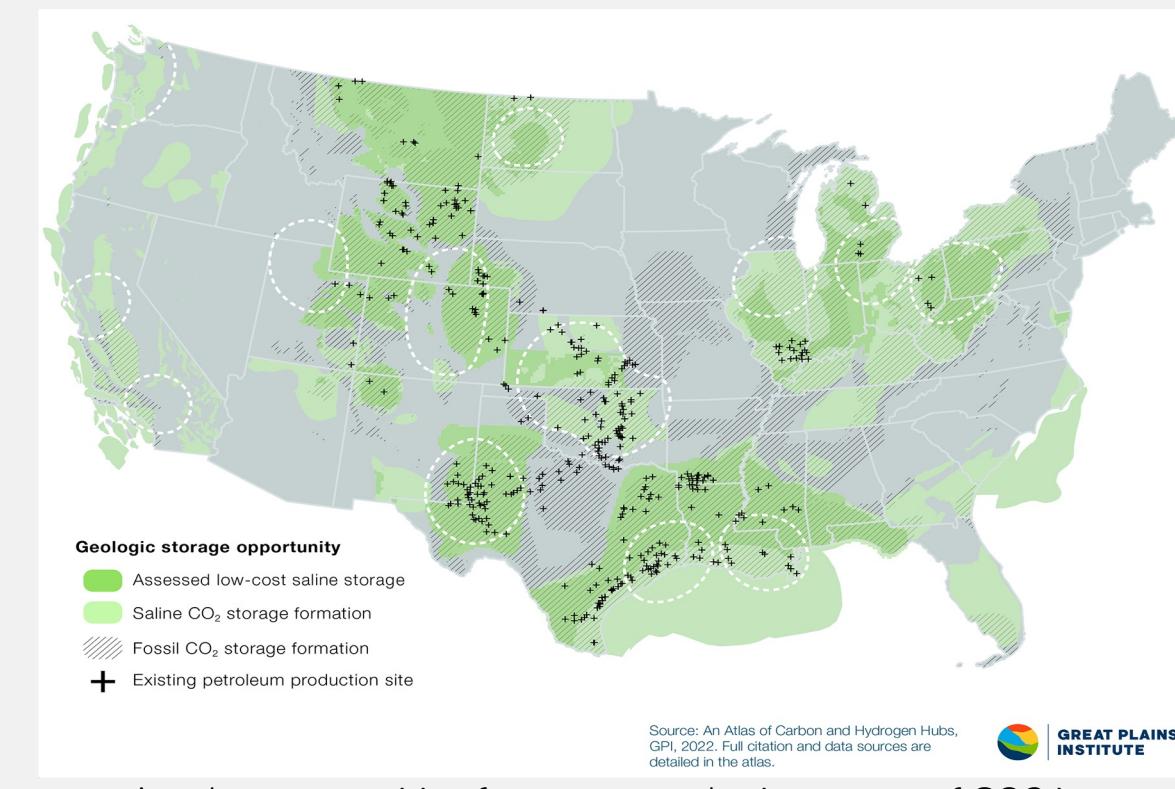
The Keeling Curve. ([NASA, 2024](#)).



World wide CCS Capacity. ([Statista, 2023](#)).

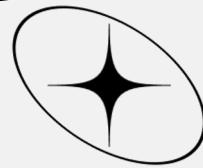


Potential for Direct Air Capture. ([Roads to Removal](#)).

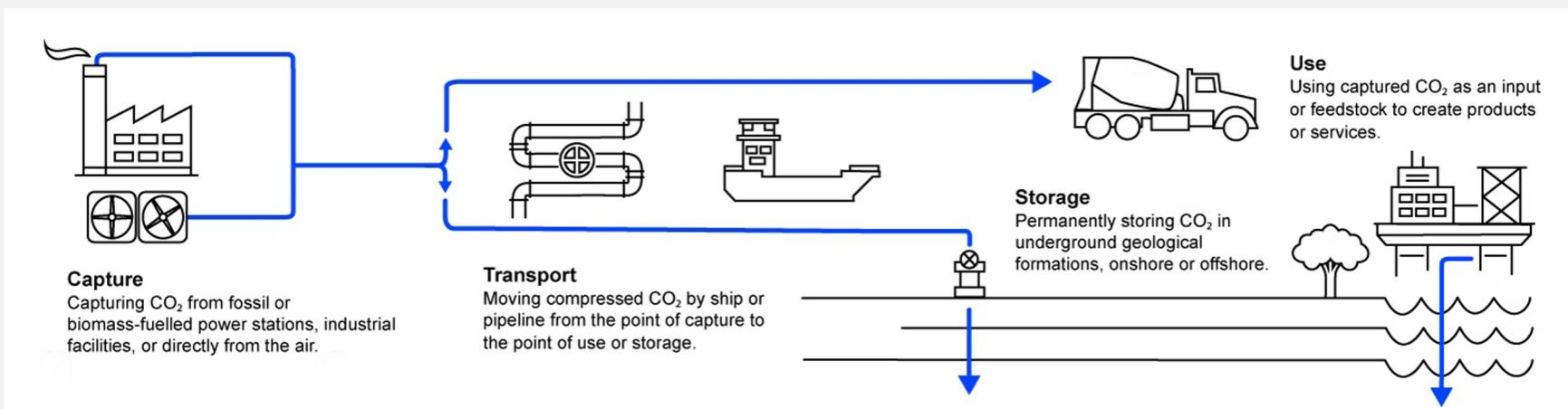


Ample opportunities for secure geologic storage of CO<sub>2</sub> in the US. ([Great Plains Institute, 2022](#)).





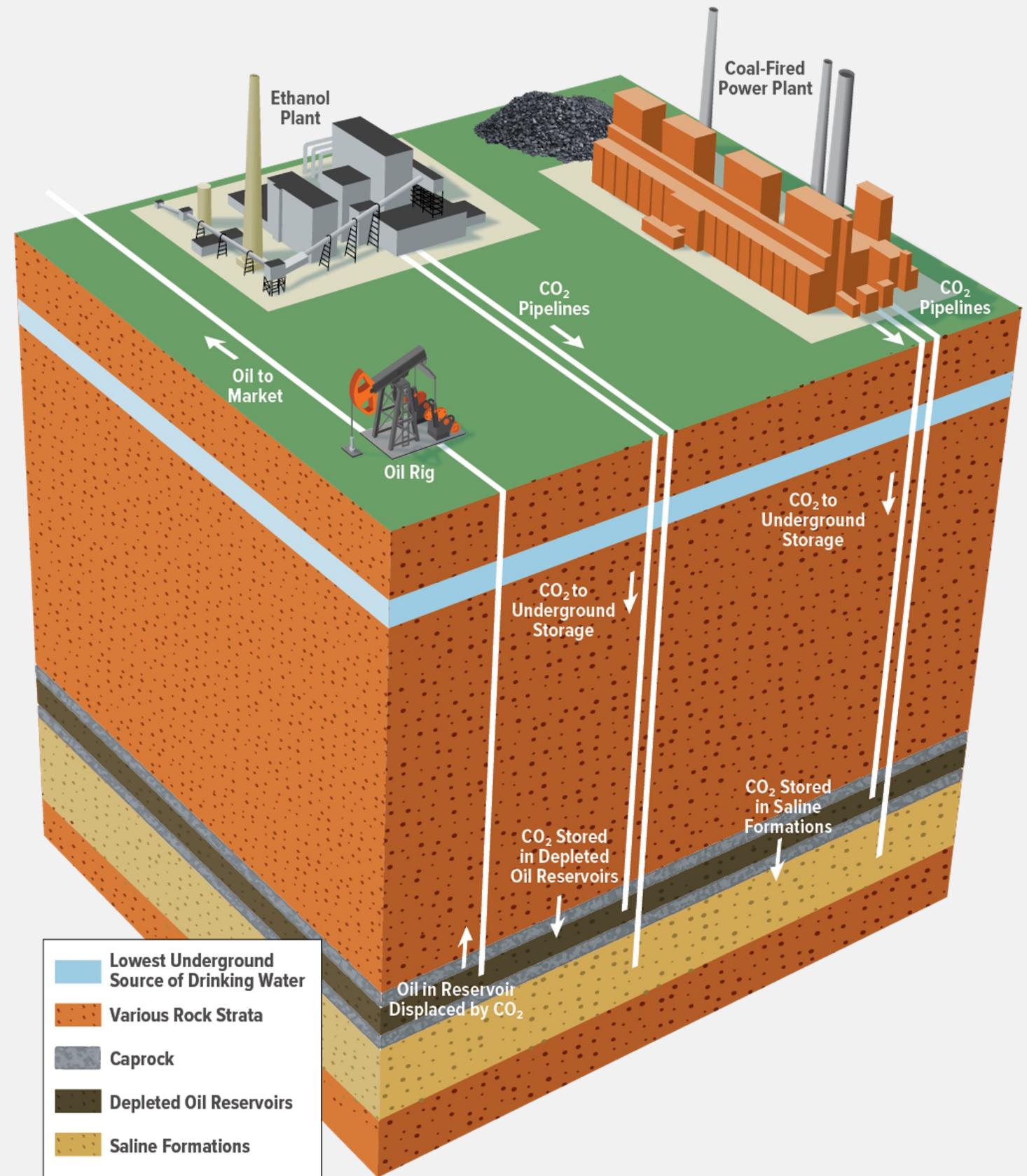
# Background



CCS Process. ([IEA](#), 2023).

United States has net-zero emission goal by 2050, 65% reduction by 2030.

Clean Energy Conversion Lab (CECL), Kleinman Center for Energy Policy, focus on CO<sub>2</sub> transportation cost & risk optimization, and repurposing railway and pipeline for CO<sub>2</sub> transport.

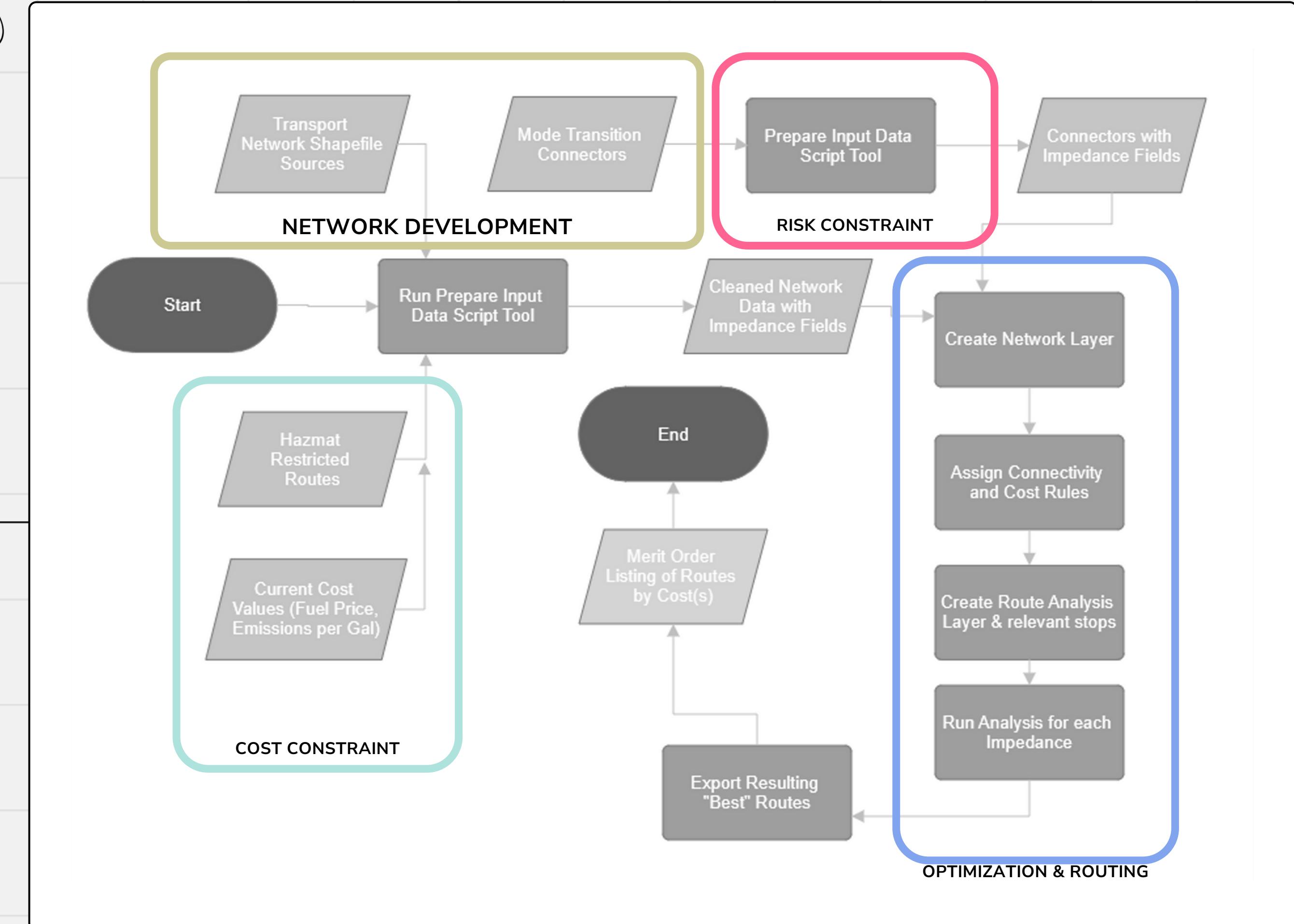


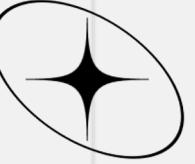
Carbon Capture and Storage Procedure Demo. (Great Plains Institute, 2022).



Our main workflow is to develop and calculate the risk and cost constraints of each road/link segment of our network, and optimize based on the constraints to minimize cost & risk.

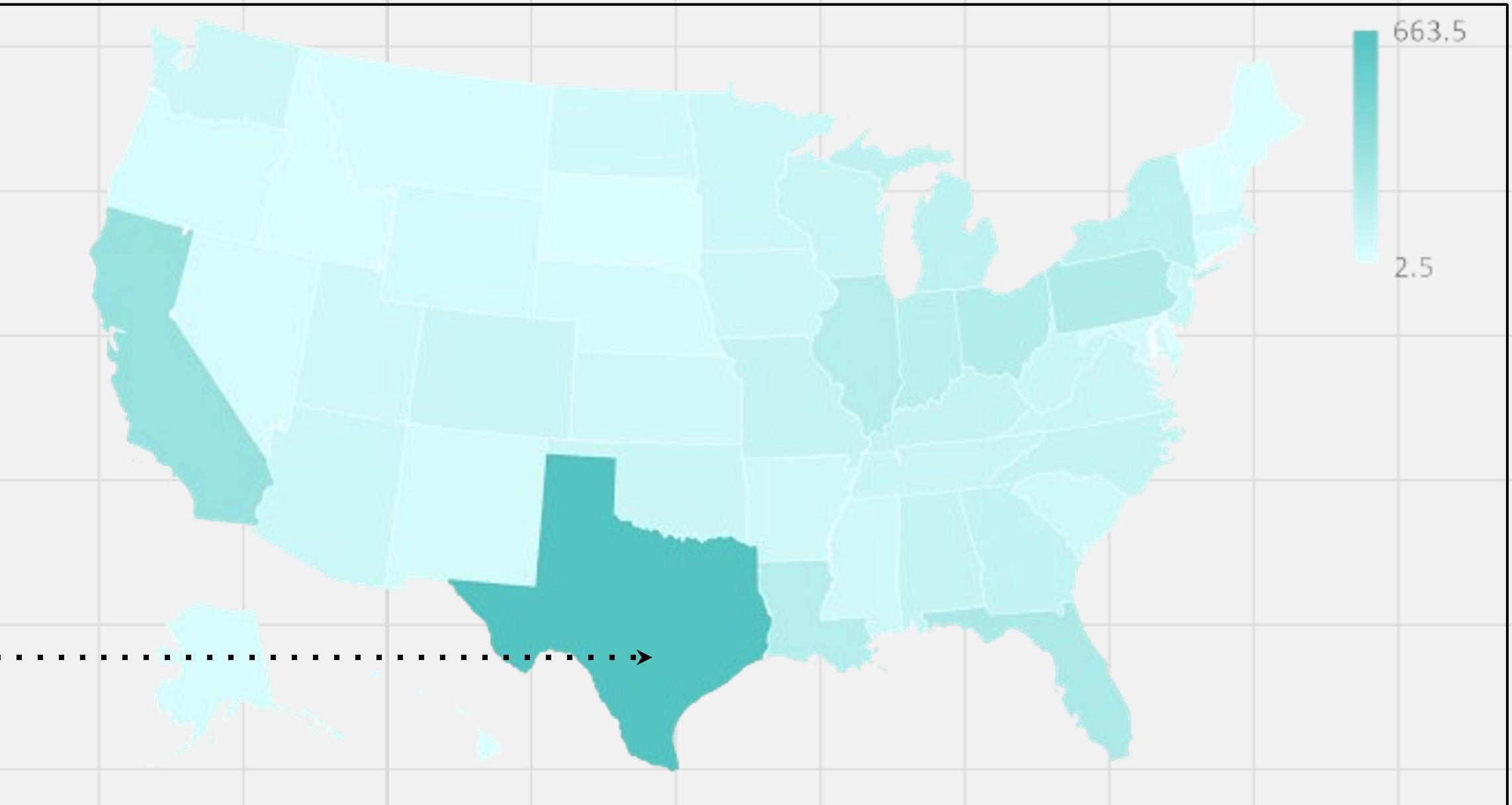
# METHODOLOGY





# STUDY AREA

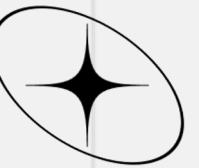
Texas is home to a diverse range of industries, including oil and gas, manufacturing, and energy production, which collectively contribute to significant CO<sub>2</sub> emissions.



CO<sub>2</sub> EMISSION BY STATE (TONS)

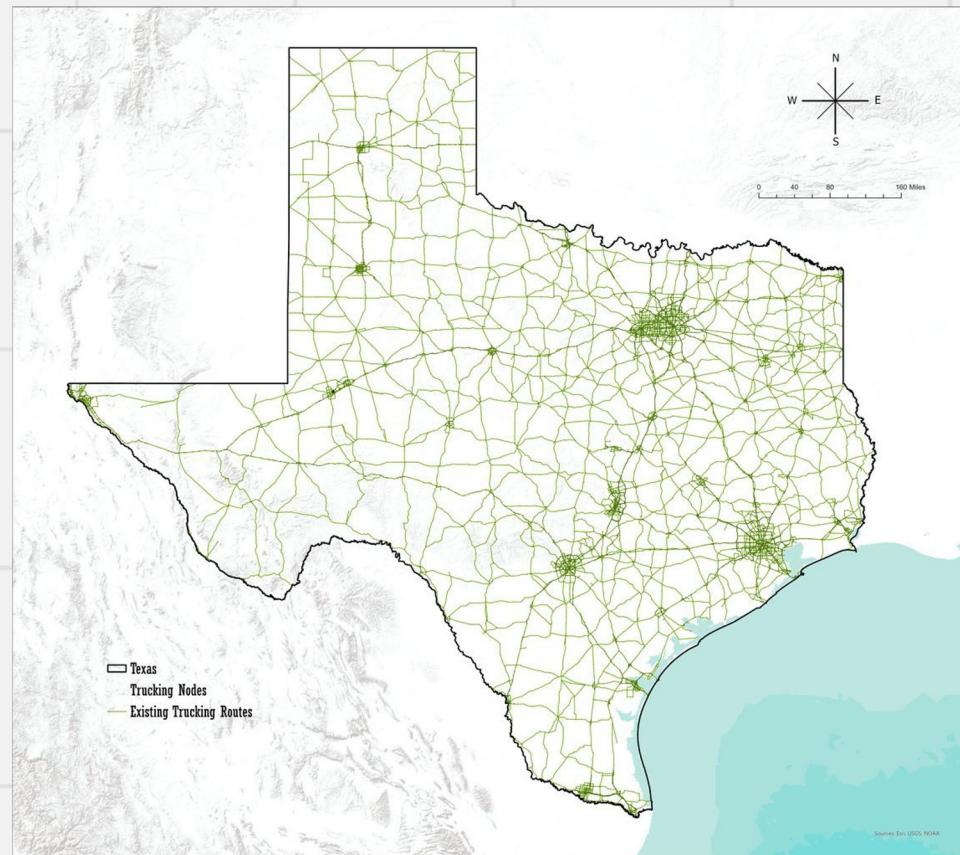


@STUDY AREA

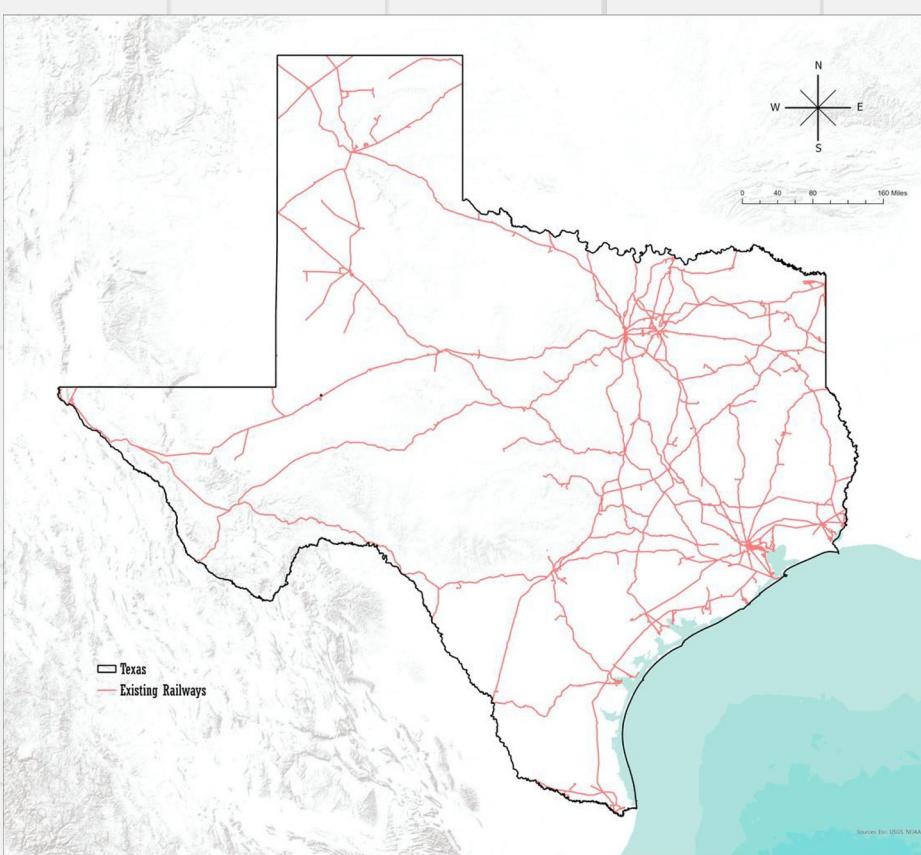


# STUDY AREA TRANSPORTATION NETWORK

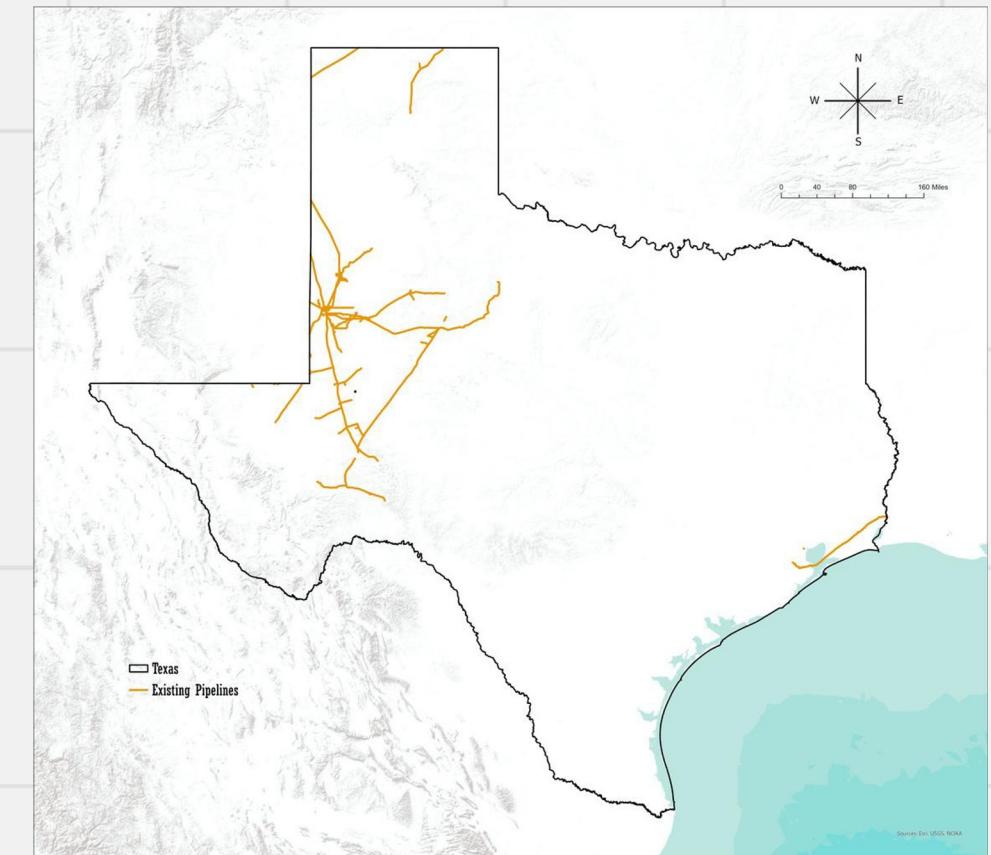
Our network model includes trucking routes, railways and pipelines



TRUCKING ROUTES

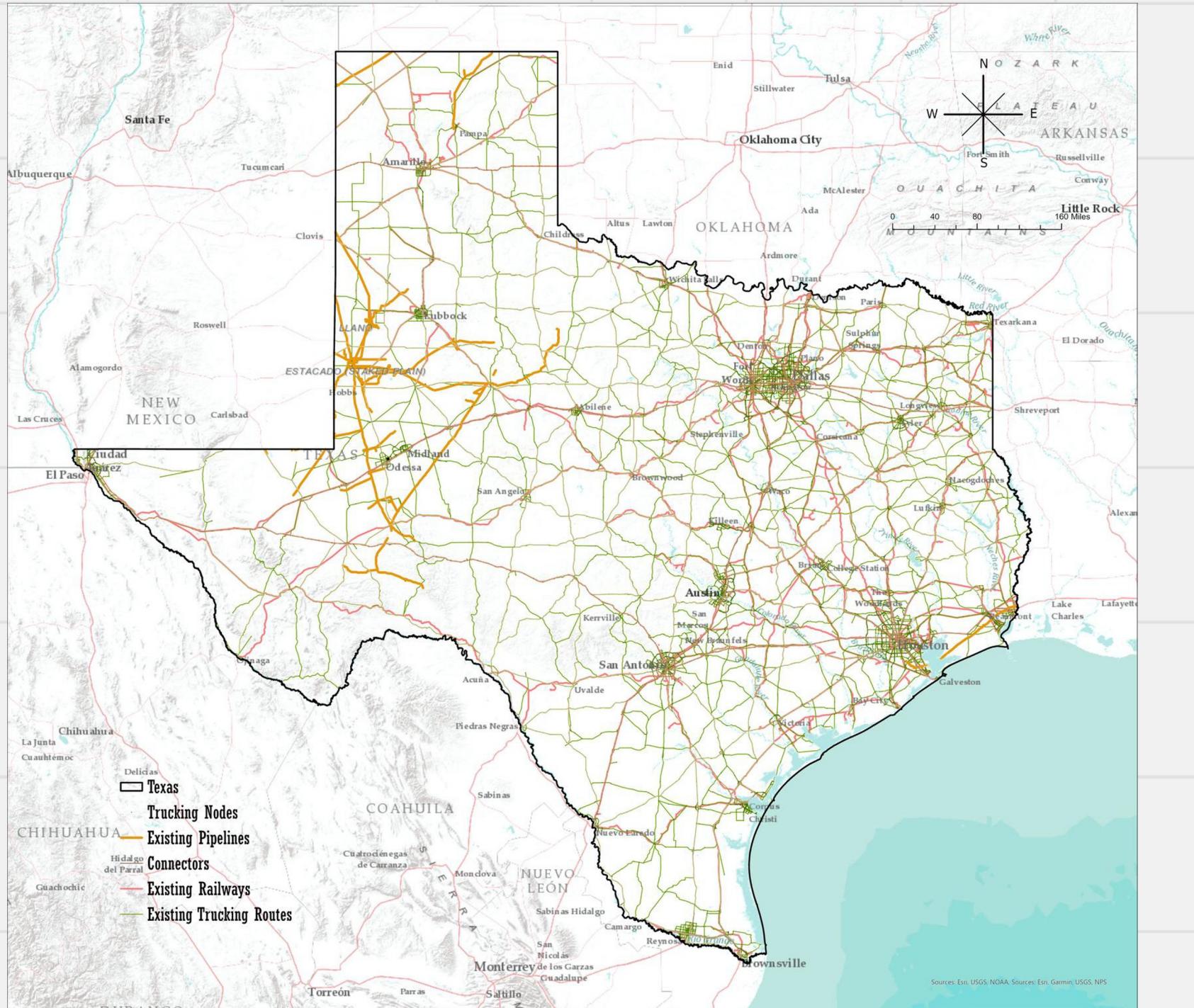


RAILWAYS



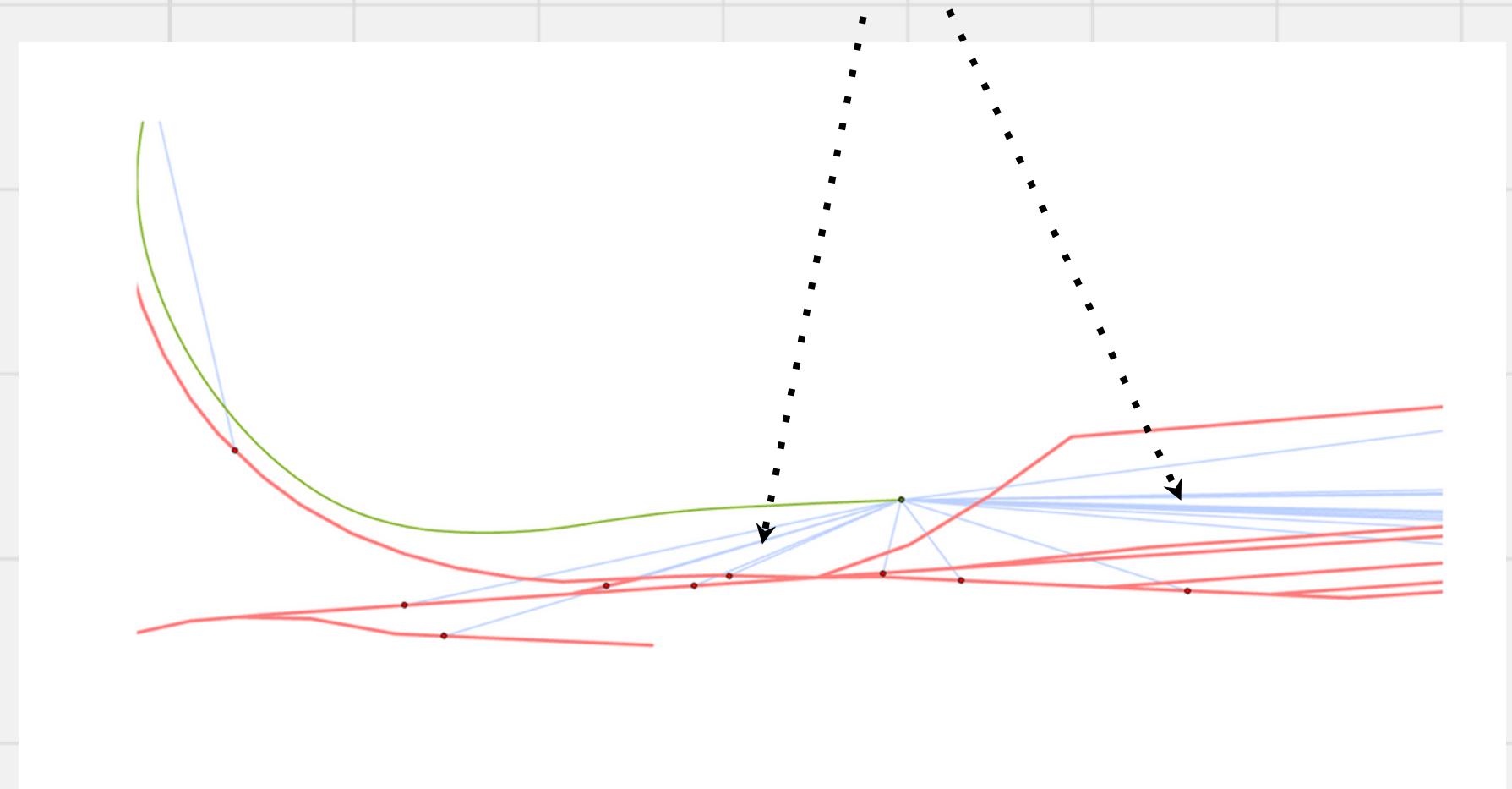
PIPELINES

@STUDY AREA



OUR MULTIMODAL TRANSPORTATION NETWORK WITH CONNECTORS BUILT

# TRANSPORTATION NETWORK - CONNECTORS

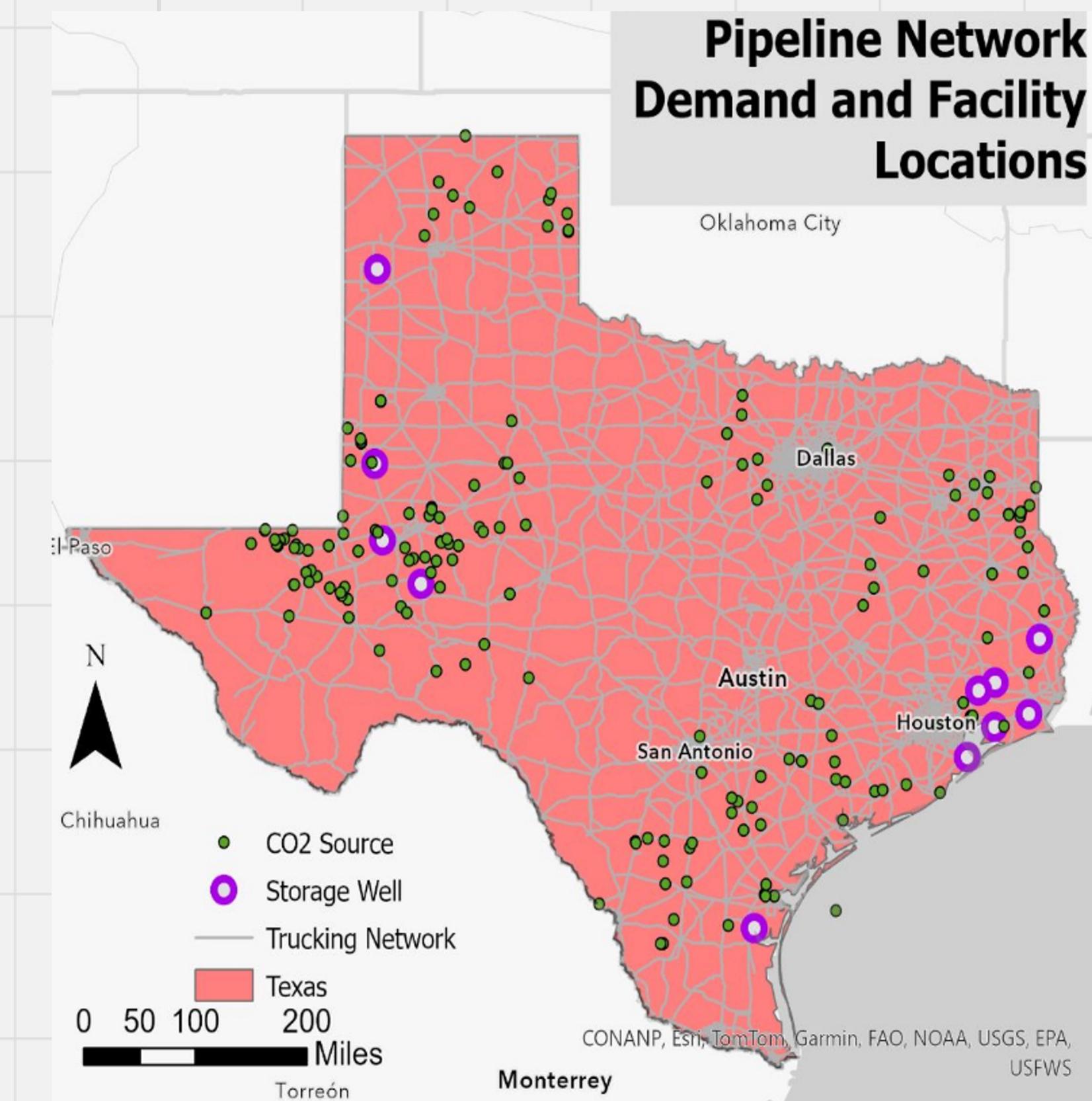


To allow transfer between rail and trucking, connectors between trucking nodes and rail yards are built using the Near Function

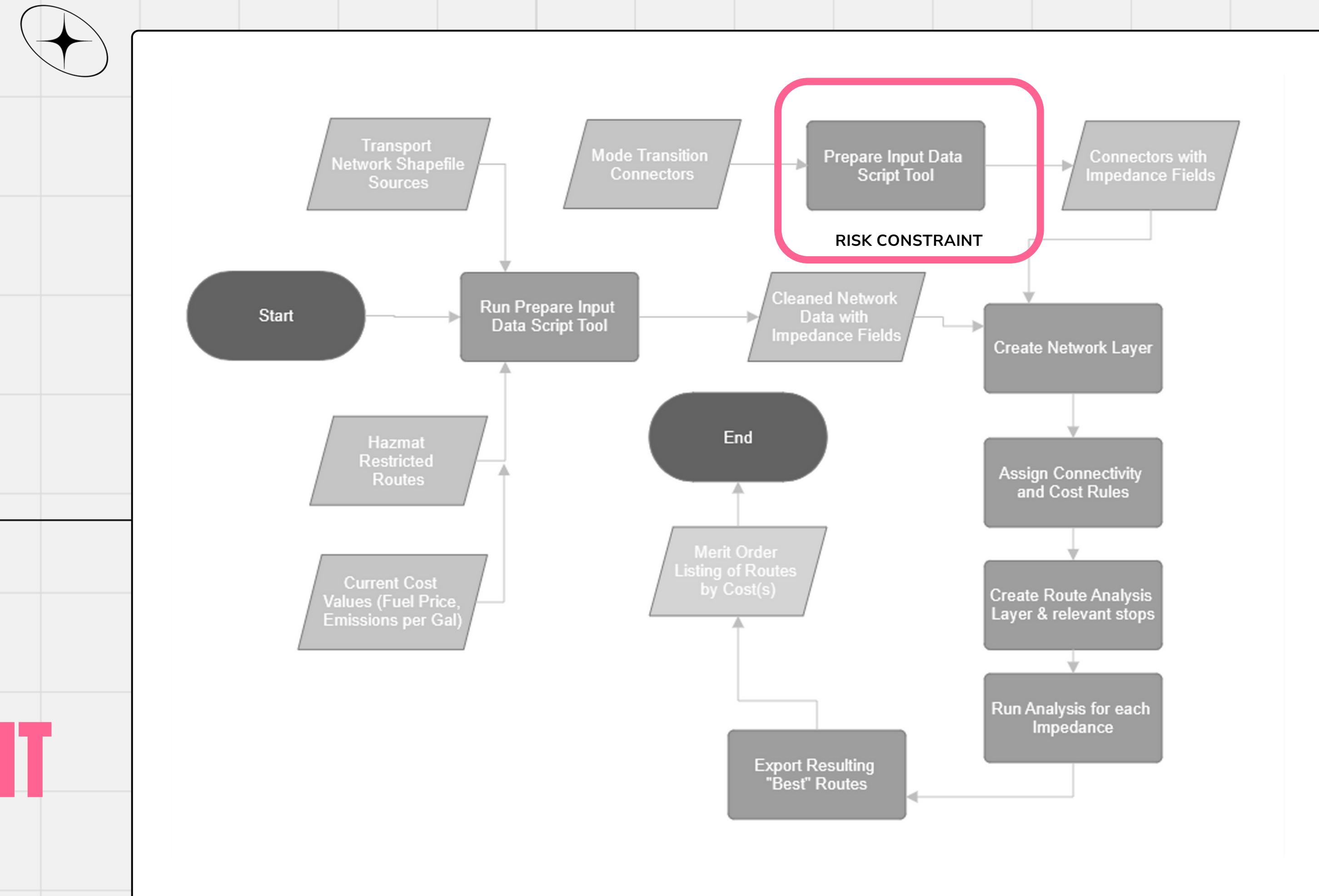
@STUDY AREA

# STUDY AREA - CO<sub>2</sub> TRANSPORT ORIGINS & DESTINATIONS

CO<sub>2</sub> SOURCE (ORIGINS) AND CARBON STORAGE FACILITIES (DESTINATIONS) IN TEXAS



# RISK CONSTRAINT





# RISK MODEL FOR PIPELINES

$X1P_i$  is the **average amount of fatalities** per incident for accident type  $P_i$

$X2P_i$  is the **average amount of injuries** per incident for accident type  $P_i$

$X3P_i$  is the **distance from the location on the pipeline with the highest probability of an incident to the closest hospital** in meters

$X4P_i$  is the **average amount of CO<sub>2</sub> released** per incident for pipeline type  $P_i$

$X5P_i$  is the **total population in a radius of 5 km of the pipeline**

$X6P_i$  is the **total number of kilometers a pipeline traverses vulnerable land**

MODEL

$$\begin{aligned}
 Risk_P &= \sum_{j=1}^2 Frequency_{P_j} \cdot Severity_{P_j} \\
 &= \sum_{i=1}^2 \sum_{k=1}^7 f_{P_i} w_{k_{P_i}} X_{k_{P_i}} \\
 &= (f_{P_1} w_{1_{P_1}} X_{1_{P_1}} + \dots + f_{P_1} w_{6_{P_1}} X_{6_{P_1}}) + (f_{P_2} w_{1_{P_2}} X_{1_{P_2}} + \dots + f_{P_2} w_{6_{P_2}} X_{6_{P_2}}).
 \end{aligned}$$



# RISK MODEL FOR MULTIMODAL-APPROVED ROUTES

X1 is the **average amount of fatalities** per incident for a road

X2 is the **average amount of injuries** per incident for a road

X3 is the **distance from the road to the closest hospital** in meters

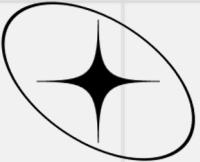
X4 is the **average amount of CO<sub>2</sub> released** per incident

X5 is the **average population density** in a radius of 1 km of the road segment

X6 is the total number of kilometers a road **traverses vulnerable land**.

MODEL

$$\begin{aligned} Risk_T &= Frequency_T \cdot Severity_T \\ &= f\vec{x}\vec{W} \\ &= fw_1X_1 + \dots, fw_6X_6. \end{aligned}$$



# Tool 1 Calculate Total Number of Population Within 5 Kilometer of Pipeline Segments

## Input:

Pipeline Shapefile

Population Raster

## Output:

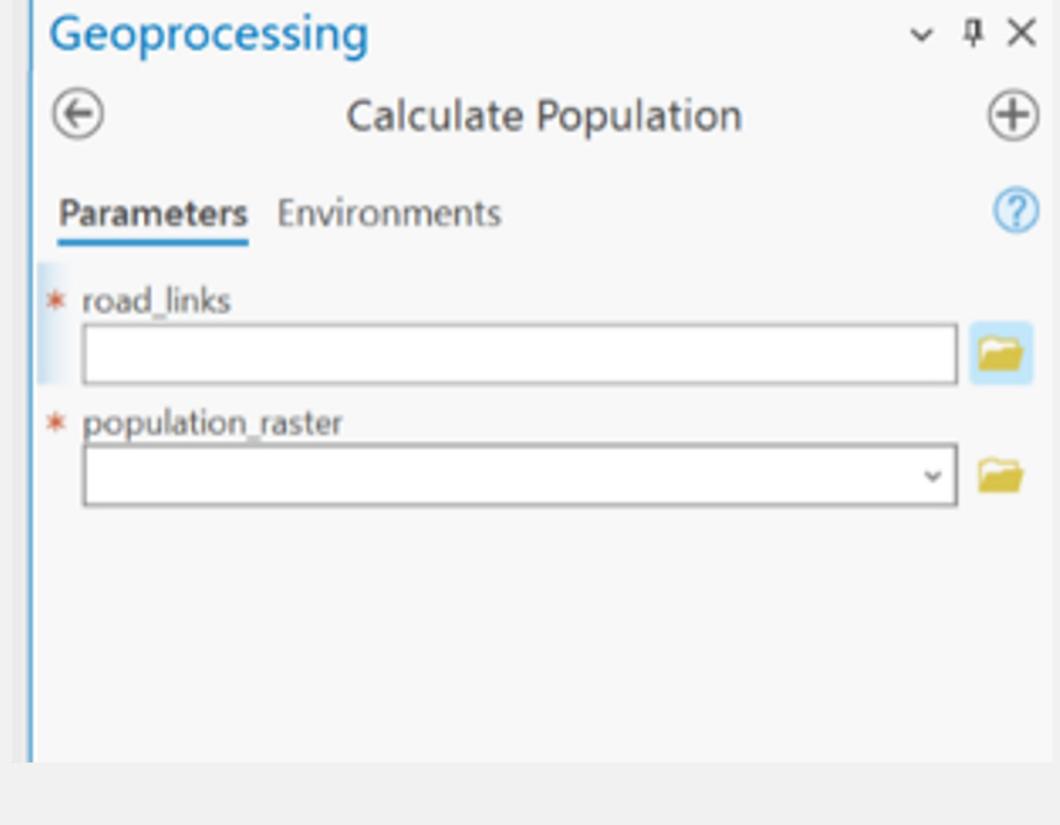
Updated pipeline Shapefile with fields holding:

1. the total number of population within 5 kilometer of the pipelines
2. The 5-kilometer radius buffered area

## PYTHON SCRIPT TOOL

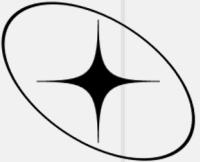
```
TRY:  
    ROAD_LINKS = ARCPY.GETPARAMETERASTEXT(0)  
    POPULATION_RASTER = ARCPY.GETPARAMETERASTEXT(1)  
    BUFFER_DISTANCE = "5000 METERS"  
  
    BUFFER_OUTPUT = "IN_MEMORY\\BUFFERED_ROADS"  
    ARCPY.BUFFER_ANALYSIS(ROAD_LINKS, BUFFER_OUTPUT, BUFFER_DISTANCE)  
  
    ARCPY.ADDFIELD_MANAGEMENT(BUFFER_OUTPUT, "AREA", "DOUBLE")  
  
    ARCPY.CALCULATEGEOMETRYATTRIBUTES_MANAGEMENT(  
        BUFFER_OUTPUT,  
        [{"AREA", "AREA_GEODESIC"}],  
        "SQUARE_METERS"  
    )  
  
    CLIPPED_RASTER = ARCPY.SA.EXTRACTBYMASK(POPULATION_RASTER, BUFFER_OUTPUT)  
  
    ZONAL_STATS_TABLE = "IN_MEMORY\\ZONAL_STATS"  
    ARCPY.SA.ZONALSTATISTICSASTABLE(BUFFER_OUTPUT, "OBJECTID", CLIPPED_RASTER,  
    ZONAL_STATS_TABLE, "DATA", "SUM")  
  
    ARCPY.JOINFIELD_MANAGEMENT(BUFFER_OUTPUT, "OBJECTID", ZONAL_STATS_TABLE,  
    "OBJECTID", ["SUM"])  
  
    ARCPY.ADDFIELD_MANAGEMENT(ROAD_LINKS, "POPULATION_DENSITY", "DOUBLE")  
  
    ARCPY.JOINFIELD_MANAGEMENT(ROAD_LINKS, "OBJECTID", BUFFER_OUTPUT,  
    "OBJECTID", ["AREA", "SUM"])  
  
    WITH ARCPY.DA.UPDATECURSOR(ROAD_LINKS, ["POPULATION_DENSITY", "SUM",  
    "AREA"]) AS CURSOR:  
        FOR ROW IN CURSOR:  
            POPULATION_SUM = ROW[1]  
            BUFFER_AREA = ROW[2]  
            POPULATION_DENSITY = POPULATION_SUM / BUFFER_AREA IF BUFFER_AREA > 0  
            ELSE 0  
            ROW[0] = POPULATION_DENSITY  
            CURSOR.UPDATEROW(ROW)  
  
    EXCEPT EXCEPTION AS E:  
        ARCPY.ADDERROR(STR(E))
```

## HOW THE TOOL LOOK LIKE IN GIS



Buffer_Area	SUM_12_13_14
0.014251	21.947788
0.036445	6349.369962
0.026829	1449.280787
0.037416	1960.007376
0.027525	1927.558027
0.016401	94.027557

## OUTPUT



# Tool 2 Calculate Total Number of Meters a Pipeline Segment Traverses Vulnerable Land

## Input:

Pipeline Shapefile

Vulnerable Area Shapefile

## Output:

Updated pipeline shapefile with  
fields holding the total number of  
meters a pipeline segment traverses  
vulnerable land

## PYTHON SCRIPT TOOL

```
IMPORT ARCPY

TRY:
    ARCPY.ENV.OVERWRITEOUTPUT = TRUE

    # INPUTS OF TRUCKS, RAILROADS, PIPES
    ROAD_LINKS = ARCPY.GETPARAMETERASTEXT(0)
    VULNERABLE_AREA_POLYGON = ARCPY.GETPARAMETERASTEXT(1)

    VULNERABLE_DISTANCE = "IN_MEMORY/VULNERABLE_DISTANCE"
    ARCPY.INTERSECT_ANALYSIS([ROAD_LINKS,
    VULNERABLE_AREA_POLYGON], VULNERABLE_DISTANCE, "ALL", "", "LINE")

    ARCPY.ADDFIELD_MANAGEMENT(VULNERABLE_DISTANCE,
    "SEGMENT_LENGTH", "DOUBLE")

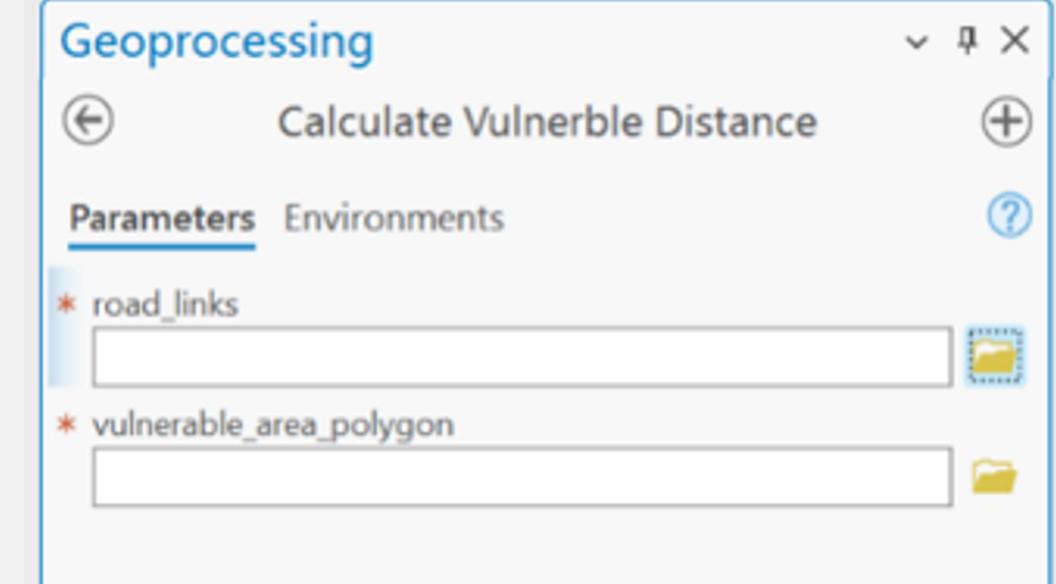
    WITH ARCPY.DA.UPDATECURSOR(VULNERABLE_DISTANCE, ["SHAPE@",
    "SEGMENT_LENGTH"]) AS CURSOR:
        FOR ROW IN CURSOR:
            ROW[1] = ROW[0].LENGTH
            CURSOR.UPDATEROW(ROW)

    ARCPY.JOINFIELD_MANAGEMENT(ROAD_LINKS, "OBJECTID",
    VULNERABLE_DISTANCE, "OBJECTID", ["SEGMENT_LENGTH"])

    ARCPY.DELETE_MANAGEMENT(VULNERABLE_DISTANCE)

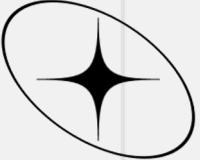
EXCEPT EXCEPTION AS E:
    ARCPY.ADDERROR(STR(E))
```

## HOW THE TOOL LOOK LIKE IN GIS



Segment_Length
27404.400234
24038.755806
23310.475617
22897.028975
20108.874094
16131.885429

## OUTPUT



# Tool 3 Calculate Average Precipitation of the Area where the Segment is Located

## Input:

Pipeline Shapefile

Precipitation Shapefile

## Output:

Updated pipeline shapefile with a field holding the average precipitation of the area where the pipeline segment is located

## PYTHON SCRIPT TOOL

```
IMPORT ARCPY
try:
    arcpy.env.overwriteOutput = True

    road_links = arcpy.GetParameterAsText(0)
    precipitation = arcpy.GetParameterAsText(1)

    precipitation_value = "in_memory/precipitation_value"
    arcpy.analysis.Identity(road_links, precipitation, precipitation_value,
    "ALL", "", "1")

    arcpy.AddField_management(road_links, "precip_avg", "DOUBLE")

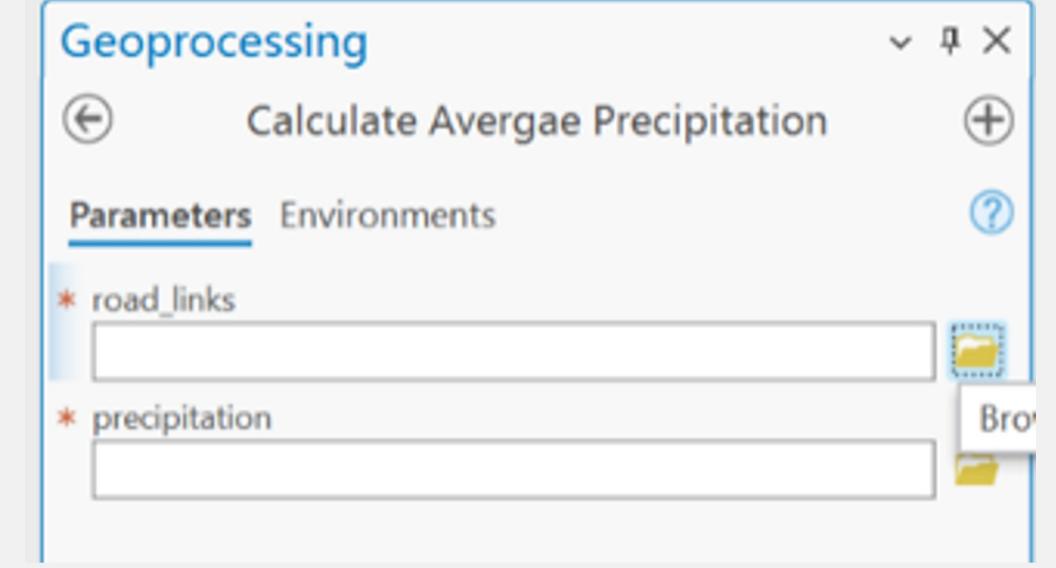
    # Store total precipitation and count of overlaps for each road link
    precipitation_totals = {}
    with arcpy.da.SearchCursor(precipitation_value, ["FID_Natura",
    "LEFT_PrecipInch"]) as cursor:
        for row in cursor:
            road_link_id = row[0]
            precip_value = row[1]
            if road_link_id not in precipitation_totals:
                precipitation_totals[road_link_id] = {'total': 0, 'count': 0}
                precipitation_totals[road_link_id]['total'] += precip_value
                precipitation_totals[road_link_id]['count'] += 1

            with arcpy.da.UpdateCursor(road_links, ["FID_Natura", "precip_avg"]) as
cursor:
                for row in cursor:
                    road_link_id = row[0]
                    if road_link_id in precipitation_totals:
                        total_precip = precipitation_totals[road_link_id]['total']
                        count = precipitation_totals[road_link_id]['count']
                        row[1] = total_precip / count if count != 0 else 0
                    else:
                        row[1] = 0
                    cursor.updateRow(row)

                arcpy.Delete_management(precipitation_value)

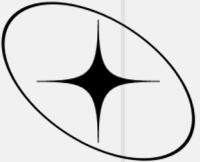
except Exception as e:
    arcpy.AddError(str(e))
```

## HOW THE TOOL LOOK LIKE IN GIS



precip_avg
57
56.5
59
12
24
34.333333

OUTPUT



# Tool 4 Calculate Distance from the Pipeline Segment to Nearest Hospital

## Input:

Pipeline/Link Shapefile

Hospital Points Shapefile

Unit of Measure

## Output:

Updated pipeline shapefile with a field holding calculated distance in the desired units

## PYTHON SCRIPT TOOL

```
IMPORT ARCPY
import os
arcpy.env.overwriteOutput = True
DEBUG = True

def getHospDist(lineFC,hosp,dist_unit,overwrite,outpath):

    # set temporary field/feature class locations
    if overwrite:
        finalFC = lineFC
    else:
        finalFC = arcpy.management.CopyFeatures(lineFC,outpath)

    if DEBUG:
        arcpy.AddMessage(f"Distance unit:{dist_unit}")

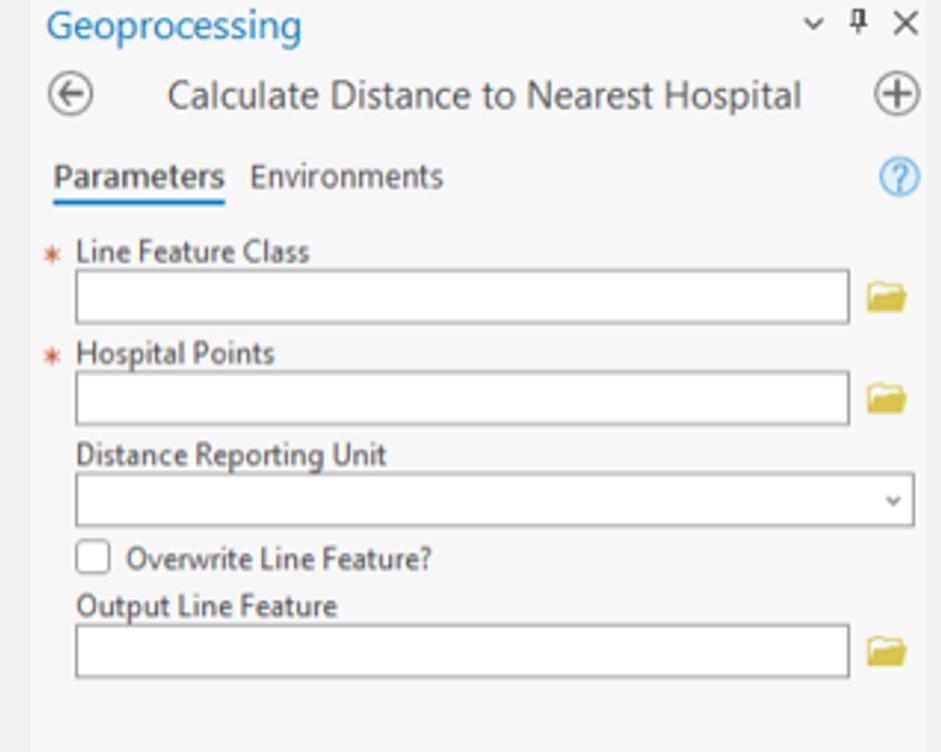
    # get near feature coordinates & set field names (saved in pointCpy)
    if dist_unit == "":
        arcpy.analysis.Near(finalFC, hosp,
                            field_names = [["NEAR_DIST","HOSP_DIST"]])
    else:
        arcpy.analysis.Near(finalFC, hosp,
                            location = False,
                            distance_unit = dist_unit,
                            field_names = [["NEAR_DIST","HOSP_DIST"]])

    if DEBUG:
        for f in arcpy.ListFields(finalFC):
            arcpy.AddMessage(f.name)
    # rename NEAR_DIST field to HOSP_DIST
    arcpy.management.DeleteField(finalFC,[ "NEAR_DIST", "NEAR_FID"])
    ## might want to add average of length from (1) nearest point,
    ## (2) closest end of line segment,
    ## & (3) furthest end of line segment
    ## for more accurate number

    # clean up arcpy objects & python variables (if necessary

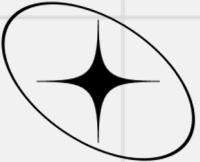
return f"Added values for distance to nearest hospital from lines."
```

## HOW THE TOOL LOOK LIKE IN GIS



HOSP_DIST
0.39163
0.389357
0.33563
0.411413
0.222993
0.38857

## OUTPUT



# Tool 5 Calculate the Number of Historic Accidents and Fatalities on a Road Link

## Input:

Route/Link Shapefile

Accidents Point Shapefile

## Output:

Updated link shapefile with a field holding the number of accidents



## PYTHON SCRIPT TOOL

```
IMPORT ARCPY

try:
    road_links = arcpy.GetParameterAsText(0)
    accidents = arcpy.GetParameterAsText(1)
    fatal_field = "Fatalists" # Replace 'Fatalists' with the actual field name in your accidents dataset that contains fatality counts

    # Create a feature layer from accidents data for spatial analysis
    accidents_layer      =      arcpy.MakeFeatureLayer_management(accidents,
"accidents_layer")

    # Add fields to store the counts of accidents and fatalities
    arcpy.AddField_management(road_links, "Total_Accidents", "DOUBLE")
    arcpy.AddField_management(road_links, "Fatalities", "DOUBLE")

    # Create buffers around road links
    buffer_distance = "50 Meters"
    buffer_output = "in_memory\\buffered_roads"
    arcpy.Buffer_analysis(road_links, buffer_output, buffer_distance)

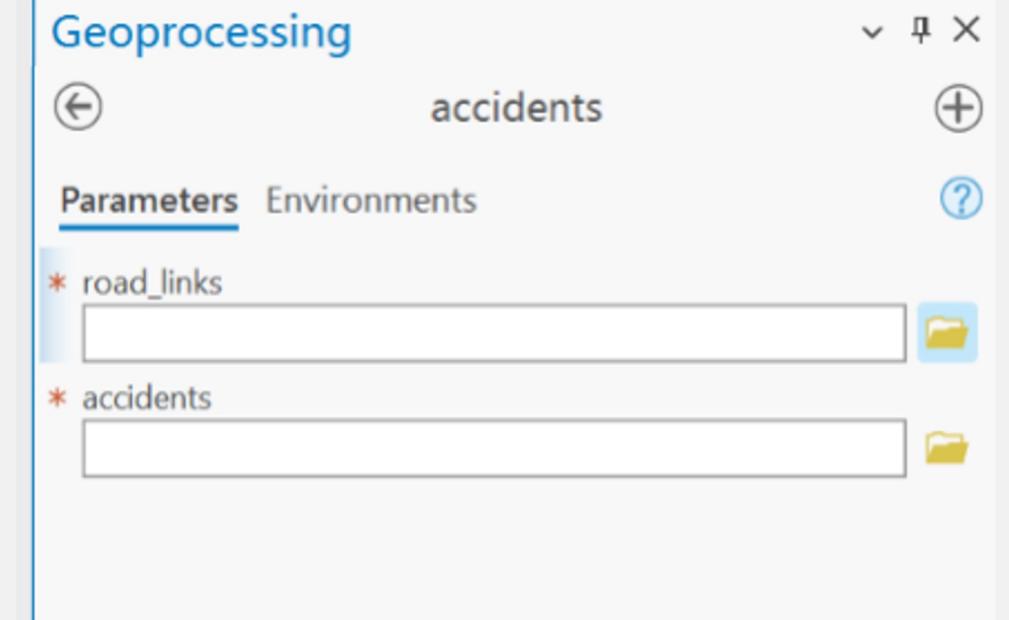
    # Intersect the buffered road links with accidents
    intersect_output = "in_memory/intersect_output"
    arcpy.Intersect_analysis([buffer_output,           accidents_layer],
intersect_output)

    # Update accident counts in the intersect_output
    arcpy.AddField_management(intersect_output, "accident_count", "DOUBLE")
    with arcpy.da.UpdateCursor(intersect_output, ["accident_count"]) as cursor:
        for row in cursor:
            row[0] = int(arcpy.GetCount_management(accidents_layer).getOutput(0))
            cursor.updateRow(row)

    # Join the calculated accident counts back to the original road links layer
    arcpy.JoinField_management(road_links,      "OBJECTID",      intersect_output,
"OBJECTID", ["accident_count"])

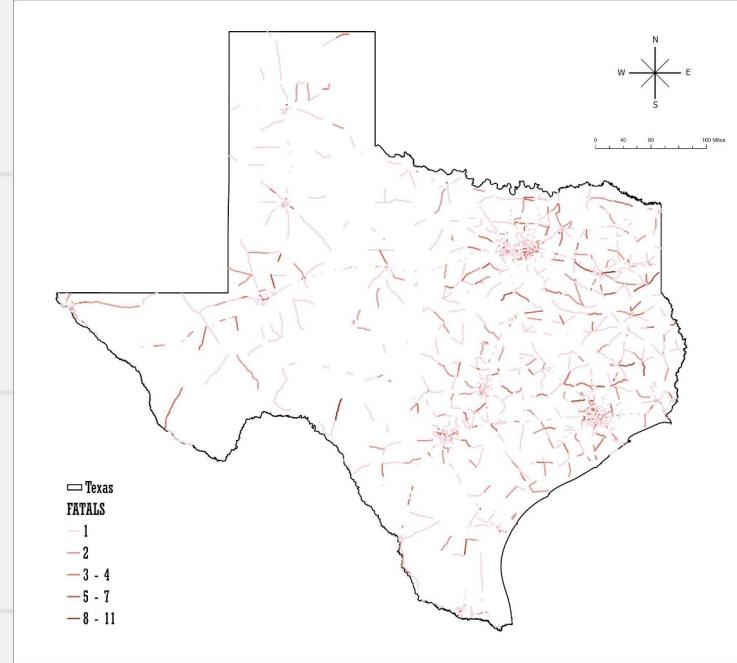
except Exception as e:
    arcpy.AddError(str(e))
```

## HOW THE TOOL LOOK LIKE IN GIS

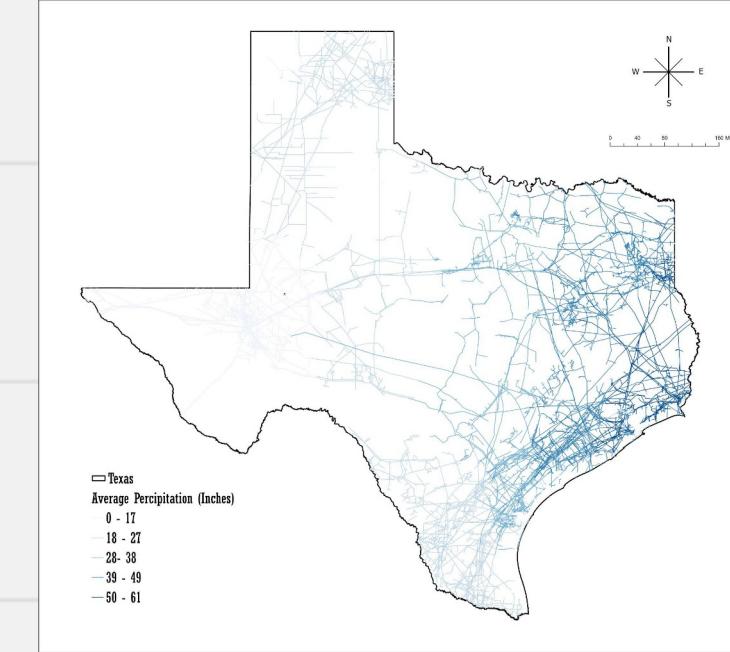


Join_Count	FATALS
7	11
2	9
7	7
7	7
5	6
3	6
1	6

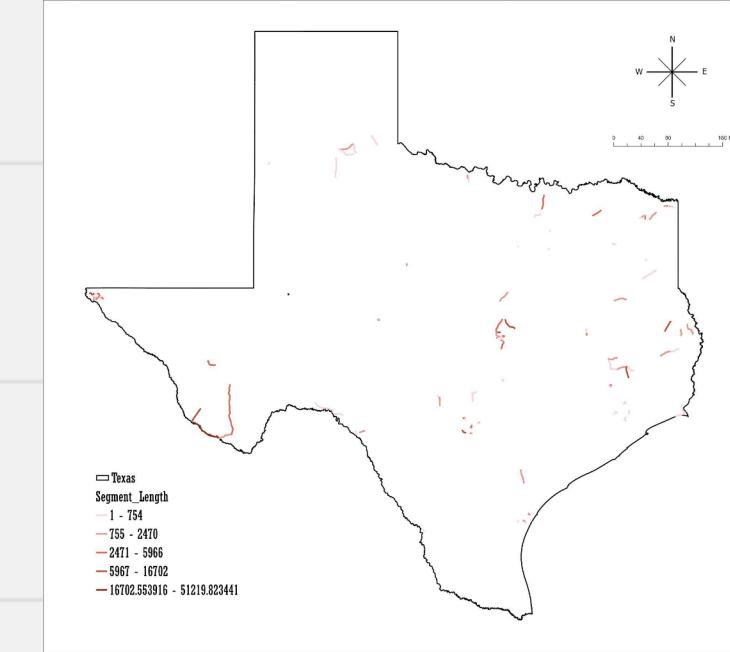
## OUTPUT



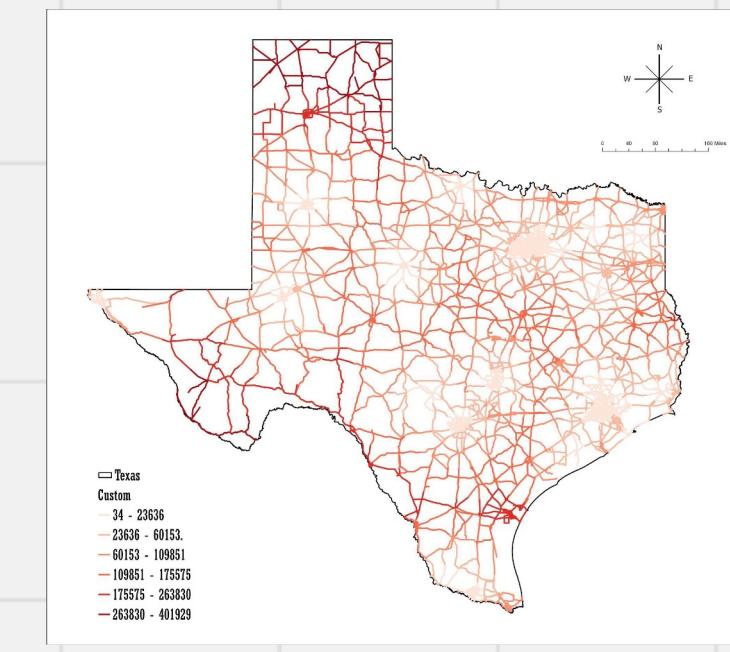
Fatalities



Average Precipitation



Vulnerable Land



Nearest Hospital Distance

Some outputs  
visualized

Scripts are developed for all modes

Calculated Variables

Risk Model

Risk Score

Segment_Length	Buffer_Area	SUM_12_13_14	precip_avg	HOSP_DIST
27404.400234	0.014251	21.947788	57	0.39163
24038.755806	0.036445	6349.369962	56.5	0.389357
23310.475617	0.026829	1449.280787	59	0.33563
22897.028975	0.037416	1960.007376	12	0.33563
20108.874094	0.027525	1927.558027	24	0.411413
16131.885429	0.016401	94.027557	34.333333	0.222993
11892.22506	0.011319	41.397817	57	0.38857
11821.152321	0.009823	0.929209	41	0.217826
10774.032192	0.007462	10.911347	45	0.222993
10545.7558	0.007435	7.857397	58.5	0.383358
9673.20835	0.009093	58.899377	54.5	0.383358
8902.835737	0.018492	73.242639	12	0.388376
8667.420818	0.052604	23.361253	55	0.374687
8615.897459	0.017858	742.279503	32.333333	0.383358
8490.677795	0.010705	1.033868	45	0.383358

$$\begin{aligned} Risk_T &= Frequency_T \cdot Severity_T \\ &= f\vec{x}\vec{W} \\ &= fw_1X_1 + \dots, fw_6X_6. \end{aligned}$$

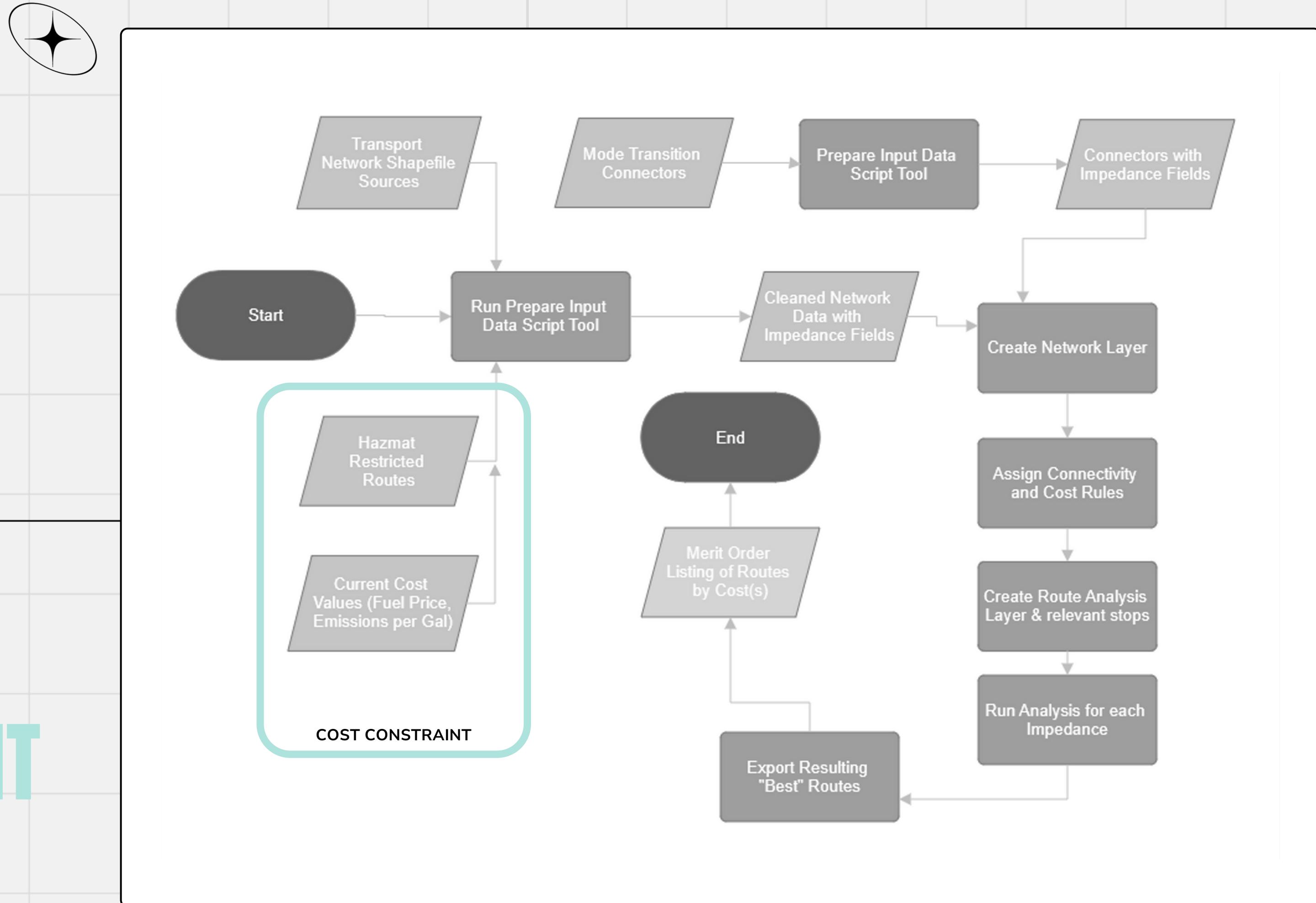
Risk_Score
0.55013
0.363179
0.365514
0.333914
0.258203
0.310667
0.784629
0.242933

# WEIGHTED RISK SCORE CALCULATION



# COST CONSTRAINT

@COST





# IMPROVED COST MODEL

## ORIGINAL COST MODEL

Previously used following variables to find Cost in \$/mile...

- Fuel cost per unit of mode
- Network edge link (considering single or round trip by mode)
- Relevant mode vehicle capacities

## UPDATED COST MODEL

- Additional time (hour) calculations
- Standardize cost to \$/mile/tonne
- Implemented
  - Pipeline mode (not multimodal)
  - Different capacities for multimodal rail cars
  - Different impedance specificities with Risk

**Geoprocessing**

Prepare Input Data

**Parameters** Environments

\* Input Features

Volume Transported

\* Mode

Multimodal Transport Possible

Number of Rail Cars

Restriction Feature

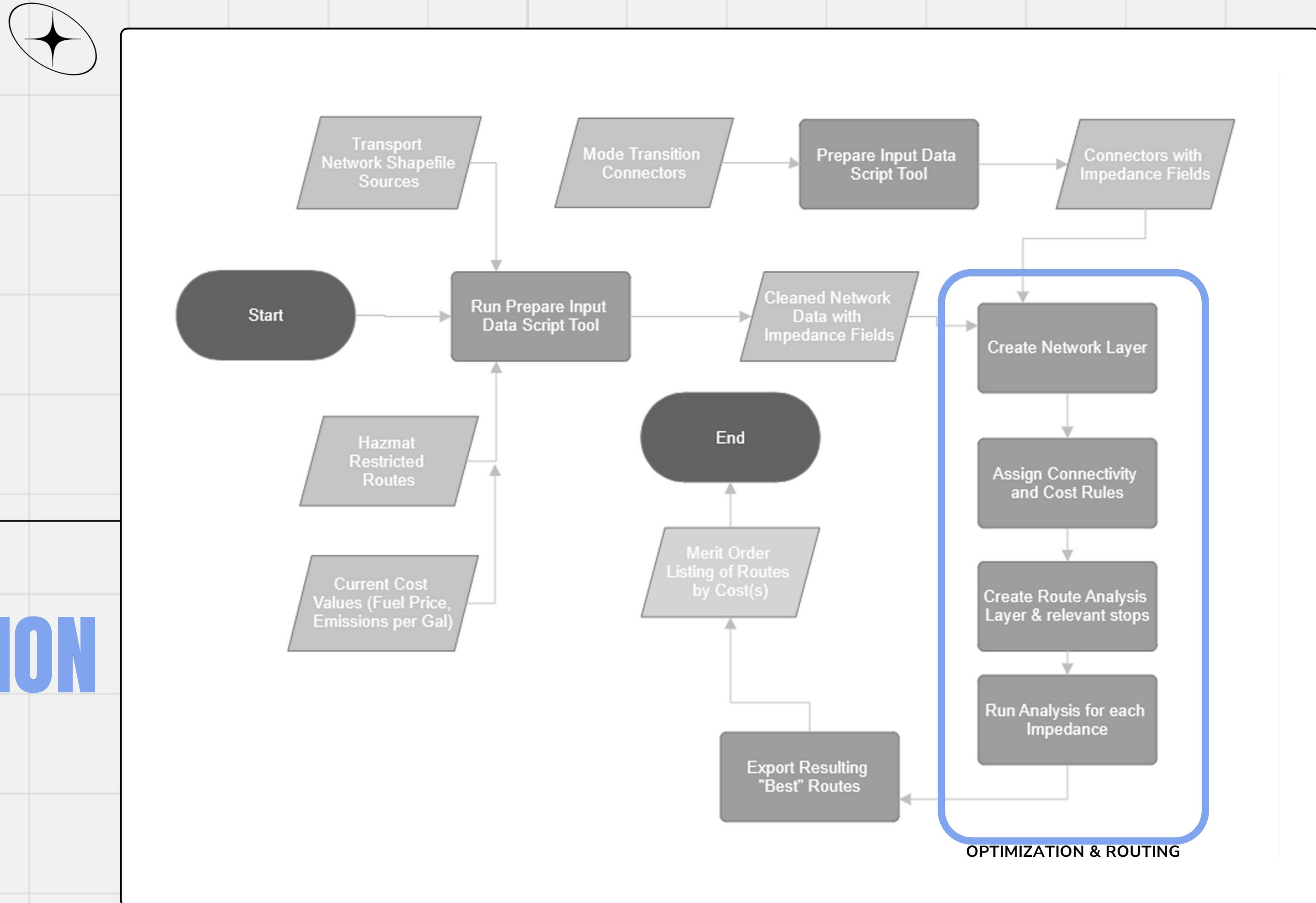
\* Spatial Reference

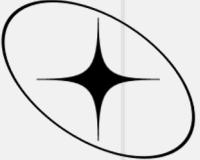
\* Output Shapefile (Folder)



# OPTIMIZATION & ROUTING

@OPTIMIZATION





# Tool 6 Create Network Dataset with the Pipeline, Railroad, Trucking Routes holding the calculated Risk Score/Cost fields

## Input:

Pipelines, railroads, trucking routes with calculated fields, connectors, nodes

## Output:

Network dataset & network analysis layer

## PYTHON SCRIPT TOOL

```
#!/usr/bin/python
# This script creates a network dataset from a feature class with calculated fields.
# It uses arcpy to interact with ArcGIS objects.
# The script takes several parameters and handles them using TryCatch blocks.

# Import arcpy module
import arcpy
from arcpy import env
def specifyParameters(args):
    return arcpy.GetParameterAsText(0)
def addParameters(args):
    for arg in args:
        print(arg.encode('utf-8'))
def handleParams():
    if arcpy.env.workspace != None:
        arcpy.AddMessage("Specifying workspace with ArcGIS Object...")
    Input_Features = arcpy.GetParameterAsText(0).split(';')
    Spatial_Reference = arcpy.GetParameterAsText(1)
    F_GDB = arcpy.GetParameterAsText(2)
    F_Database = arcpy.GetParameterAsText(3)
    Created_FC_Names = arcpy.GetParameterAsText(4).split(';')
    if len(Created_FC_Names) == 2:
        arcpy.AddMessage("Creating Feature Class Names")
    Network_Dataset = arcpy.GetParameterAsText(5)
    Overwrite_Output = arcpy.GetParameterAsText(6)
    arcpy.AddMessage("Overwrite Output: " + Overwrite_Output)
    arcpy.AddMessage("Input Features: " + Input_Features)
    arcpy.AddMessage("Spatial Reference: " + Spatial_Reference)
    arcpy.AddMessage("File Geodatabase: " + F_GDB)
    arcpy.AddMessage("Feature Class Name(s): " + Created_FC_Names)
    arcpy.AddMessage("Network Dataset Name: " + Network_Dataset)
    arcpy.AddMessage("Overwrite Output: " + Overwrite_Output)
    arcpy.AddMessage("Using present parameter...")

    Input_Features = [Input_Features]
    Spatial_Reference = [Spatial_Reference]
    F_GDB = [F_GDB]
    F_Database = [F_Database]
    Created_FC_Names = [Created_FC_Names]
    Network_Dataset = [Network_Dataset]
    Overwrite_Output = [Overwrite_Output]

    arcpy.env.workspace = F_GDB
    arcpy.env.scratchWorkspace = Input_Features[0]
    arcpy.env.scratchEnvironment = Spatial_Reference[0]
    arcpy.env.outputWorkspace = F_GDB
    arcpy.env.outputEnvironment = Spatial_Reference[0]
    arcpy.env.outputDatabase = F_Database[0]
    arcpy.env.outputFeatureDatasetName = Created_FC_Names[0]
    arcpy.env.outputNetworkDatasetName = Network_Dataset[0]
    arcpy.env.outputOverwrite = Overwrite_Output[0]

    arcpy.env.overwriteOutput = Overwrite_Output[0]
    arcpy.env.workspace = F_GDB
    arcpy.env.scratchWorkspace = Input_Features[0]
    arcpy.env.scratchEnvironment = Spatial_Reference[0]
    arcpy.env.outputWorkspace = F_GDB
    arcpy.env.outputEnvironment = Spatial_Reference[0]
    arcpy.env.outputDatabase = F_Database[0]
    arcpy.env.outputFeatureDatasetName = Created_FC_Names[0]
    arcpy.env.outputNetworkDatasetName = Network_Dataset[0]
    arcpy.env.outputOverwrite = Overwrite_Output[0]

    arcpy.management.CreateFileGDB(gdbPath, gdbName)
    try:
        arcpy.management.CreateFeatureDataset(parallelList["F_GDB"][-1].gdb,
                                             parallelList["F_Database"],
                                             parallelList["Created_FC_Names"],
                                             parallelList["Network_Dataset"])
    except:
        arcpy.management.CreateFeatureDataset(parallelList["F_GDB"][-1].gdb,
                                             parallelList["F_Database"])

    def reportFeatures(parallelList):
        names = parallelList["Created_FC_Names"]
        inputs = parallelList["Input_Features"]
        defFolder = False
        created_fcs = []
        arcpy.conversion.FeatureClassToFeatureDataset(inputs,
                                                       os.path.join(parallelList["F_GDB"][-1].gdb,
                                                       parallelList["F_Database"]))
        for name in names:
            if 'FC' in name:
                name = name.replace('FC', '')
            created_fcs.append(name)
        return created_fcs

    def reportConnections(arcs_FCs):
        fdPath = os.path.join(
            parallelList["F_GDB"][-1].gdb,
            parallelList["F_Database"])
        parallelList["F_Database"]

        print("Feature Dataset Path: ", fdPath)
        print("Source Feature Classes: ", arcs_FCs)

    try:
        arcpy.management.CreateNetworkDataset(
            "FeatureDataset",
            "Network_Dataset",
            out_name=parallelList["Network_Dataset"],
            source_feature_class=sourceFc,
            source_node_id_field="AC_ELEVATION"
        )
    except arcpy.ExecuteError as e:
        print(e.message)
        print(arcpy.GetMessages())
    print("Network Dataset creation completed.")

    def emptyText(inp, param = "unknown parameter"):
        try:
            inp()
        except ValueError as e:
            arcpy.AddMessage("Given parameter (%s) for '%s' not castable to numeric type. Parameter is set to 0." % (param, param))
            inp = 0
        return inp

    General_Notes
    # To allow overwriting outputs change overwriteOutput option to True.
    # or implement checkbox parameter in tool
    arcpy.env.overwriteOutput=True
    if arcpy.env.workspace != None:
        arcpy.AddMessage("Specifying workspace with ArcGIS Object...")
    if arcpy.env.scratchEnvironment != None:
        arcpy.AddMessage("Specifying scratch environment with ArcGIS Object...")
    if arcpy.env.outputEnvironment != None:
        arcpy.AddMessage("Specifying output environment with ArcGIS Object...")
    if arcpy.env.outputDatabase != None:
        arcpy.AddMessage("Specifying output database with ArcGIS Object...")
    if arcpy.env.outputFeatureDatasetName != None:
        arcpy.AddMessage("Specifying output feature dataset name with ArcGIS Object...")
    if arcpy.env.outputNetworkDatasetName != None:
        arcpy.AddMessage("Specifying output network dataset name with ArcGIS Object...")
    if arcpy.env.outputOverwrite != None:
        arcpy.AddMessage("Specifying output overwrite with ArcGIS Object...")

    arcpy.AddMessage("Creating File Geodatabase & Feature Dataset...")
    createdObjs=arcpy.CreateFileGDB(F_GDB, "new.gdb")
    arcpy.AddMessage("File successfully created.")

    # Export Feature Classes; create network dataset
    arcpy.AddMessage("Exporting Feature Classes & Creating Network Dataset...")
    createdFc=arcpy.FeatureClassToFeatureDataset_conversion(parallelList["Input_Features"], parallelList["Created_FC_Names"])
    arcpy.AddMessage("Feature classes successfully exported to Network Dataset successfully created.")

    arcpy.AddMessage("Feature classes successfully exported to Network Dataset successfully created.")
```

## HOW THE TOOL LOOK LIKE IN GIS

### Geoprocessing

#### Create Network Layer with TryCatch

##### Parameters Environments

##### \* Input Features

##### Spatial Reference

##### \* File GeoDatabase

##### \* Feature Dataset (name only)

##### Member Feature Class Name(s)

##### \* Network Dataset Name

##### Overwrite Output?

### dataset

#### connectors\_ExportFeatures2



#### new

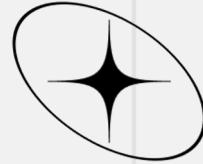
#### new\_Junctions

#### raillinks

#### truckings

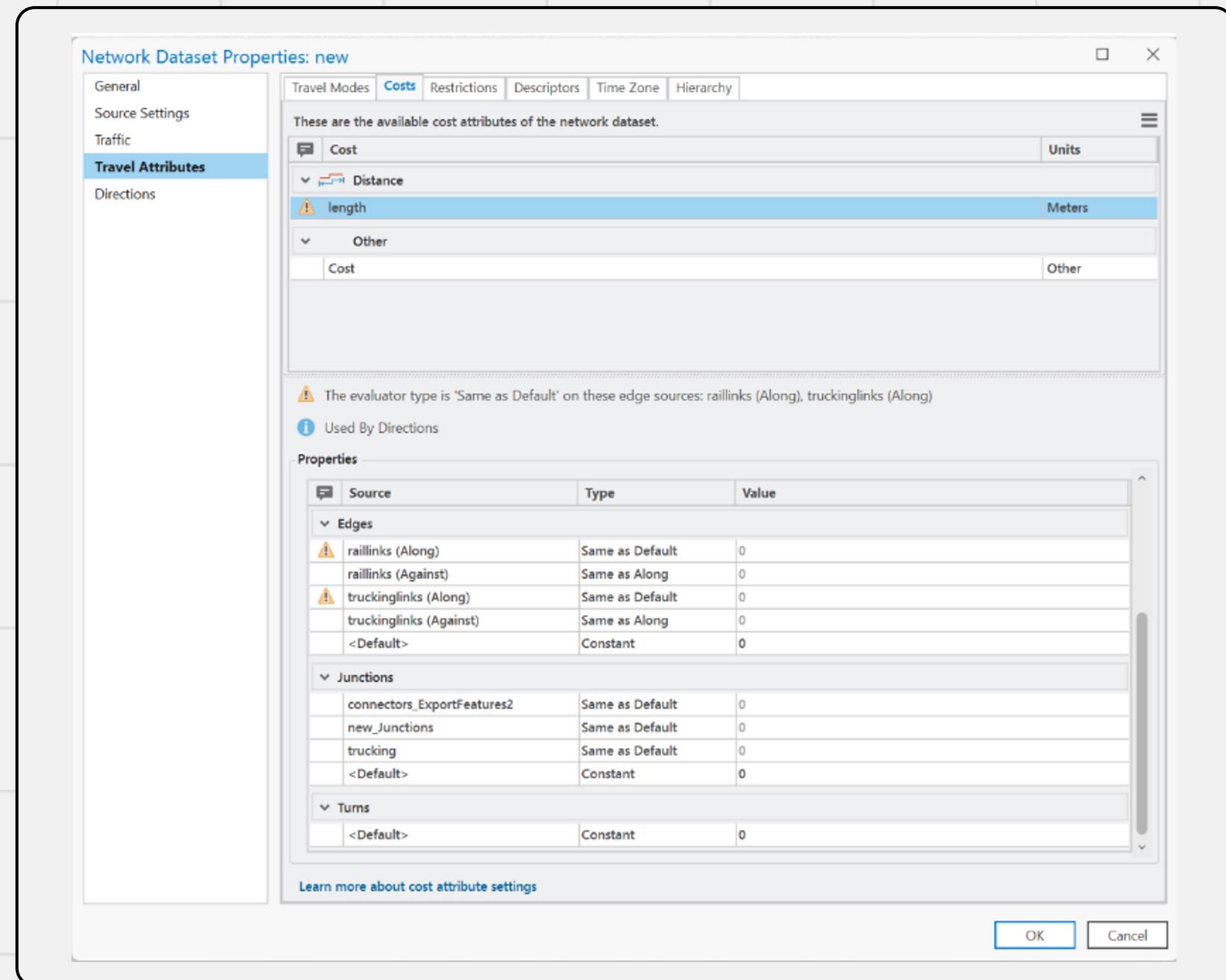
#### truckingslinks

## OUTPUT

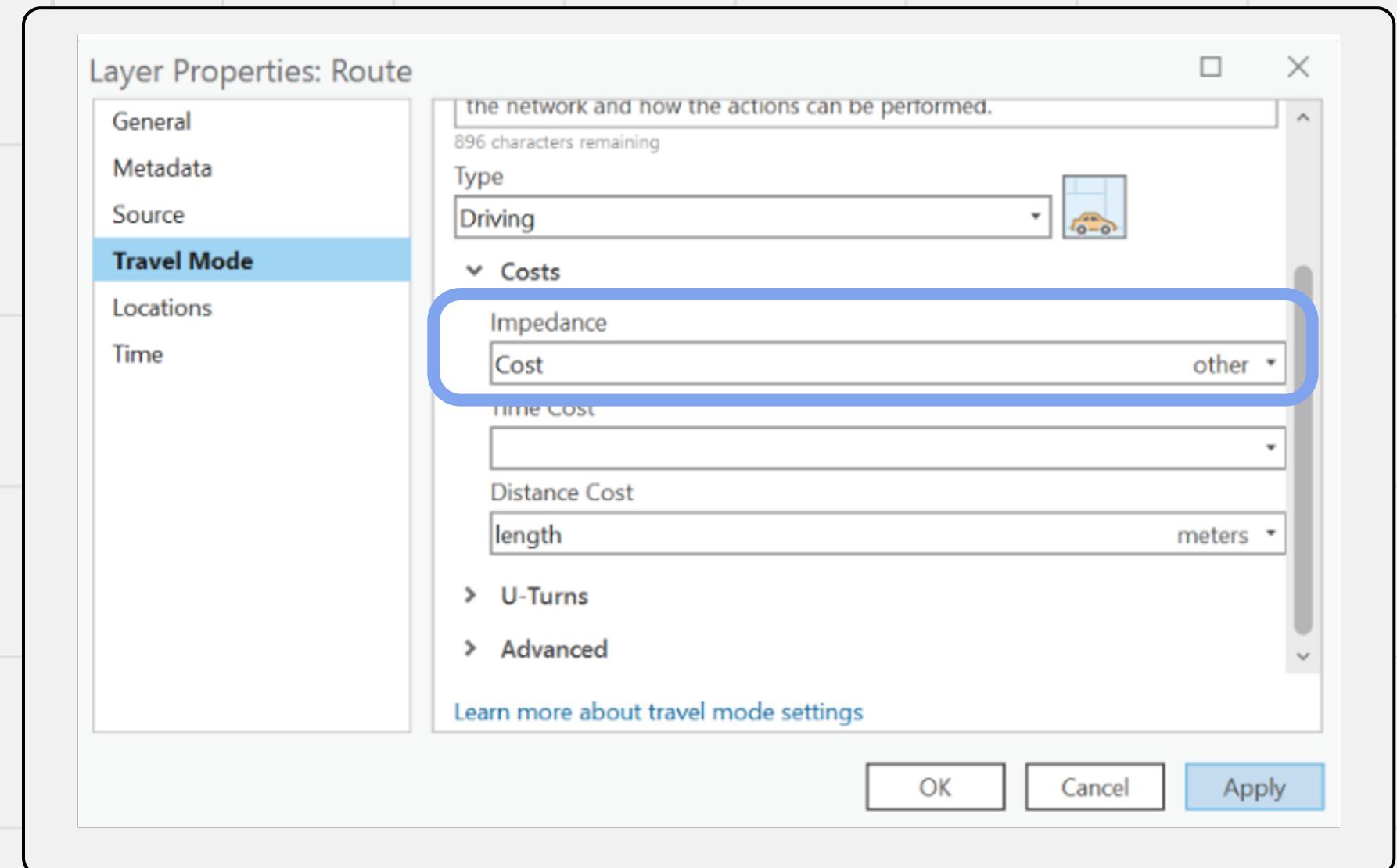


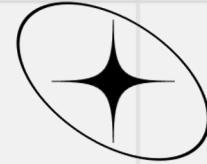
# SET UP NETWORK ANALYSIS LAYER (MANUAL STEP)

Set up network analysis properties



Change Impedance:  
Optimize Cost or Risk\_Score

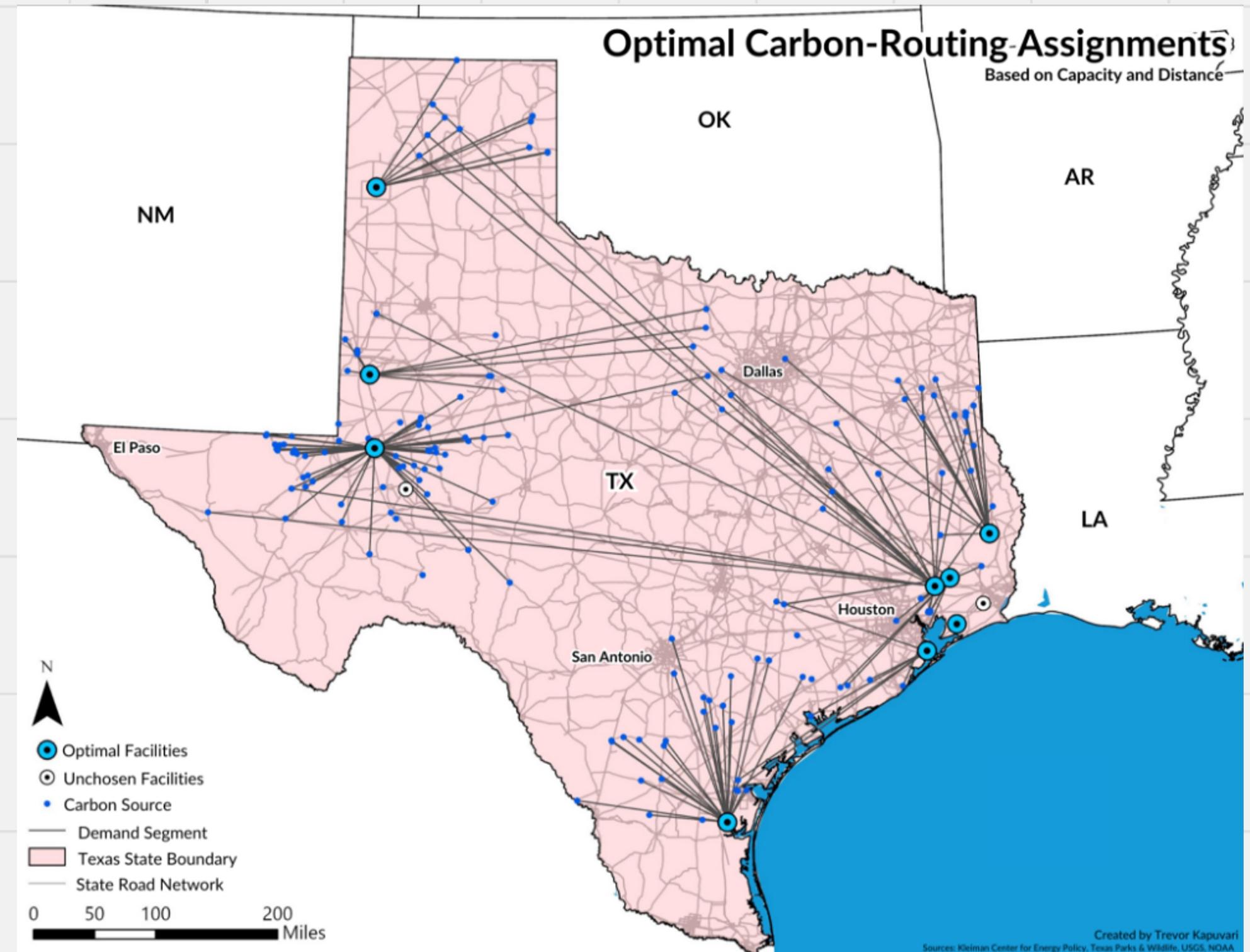


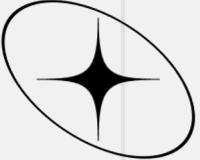


# FINDING OPTIMIZED ORIGIN-DESTINATION PAIRS

If the origin (CO<sub>2</sub> emission sources) and destination (CO<sub>2</sub> storage facilities) data are not already paired, an additional step is necessary to optimize these pairings based on the carbon storage capacity of the facilities. This is essential because origins and destinations must be paired to conduct network routing optimization effectively.

*This od-pair optimization is conducted using CPLEX:  
Minimize Travel Distance*





# Tool 7 Input Origin-destination pairs, Solve, Output the Optimized Route and Breakdown Table

## **Input:**

origin-destination pairs network  
analysis layer

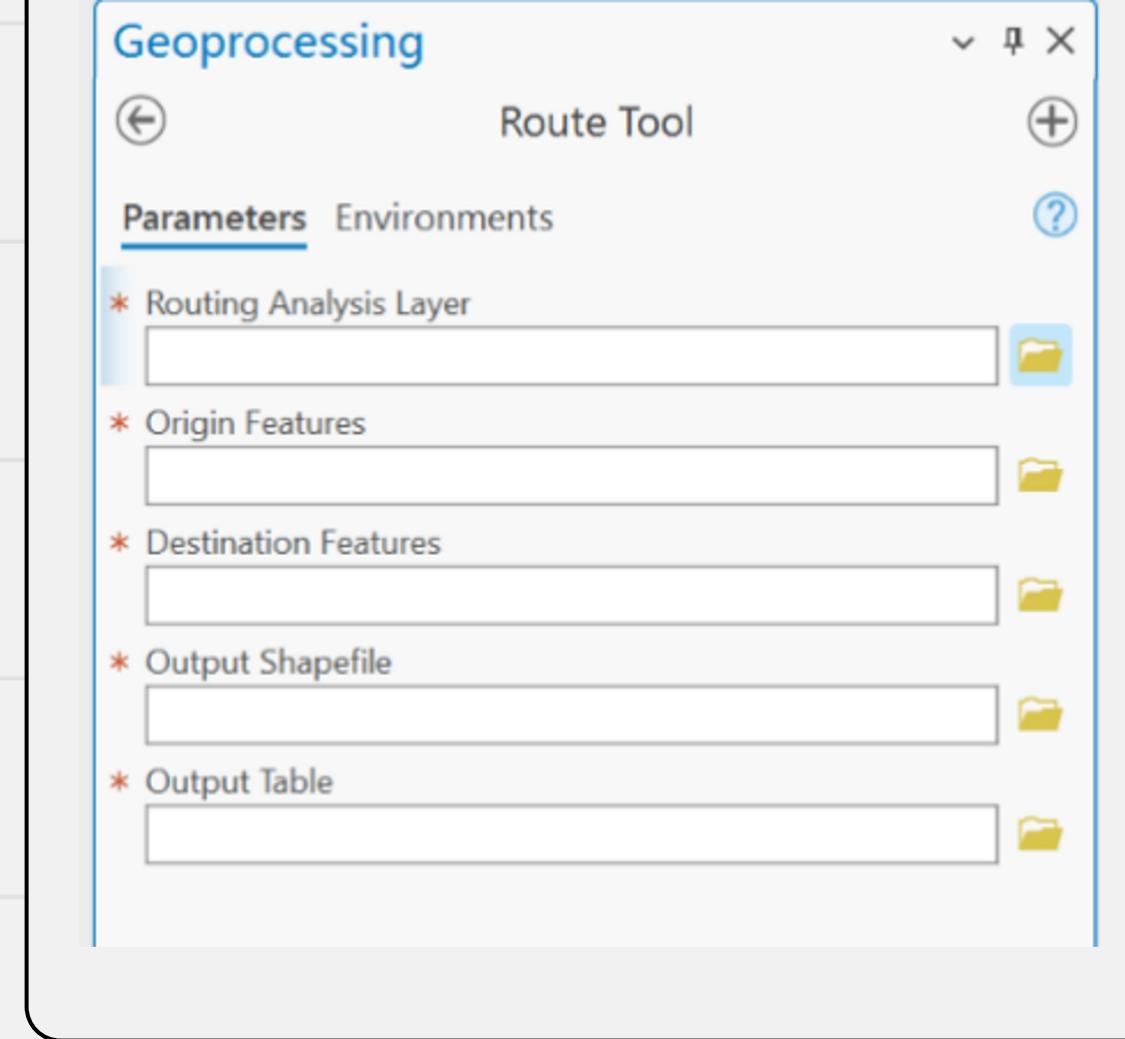
## **Output:**

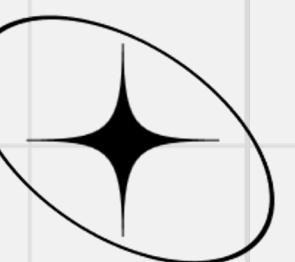
1. Visualization of the chosen route
2. Cost/risk breakdown table of the chosen routes

PYTHON SCRIPT TOOL

```
IMPORT arcpy
IMPORT os
# Import modules
# Set workspace
arcpy.env.workspace = "C:\Users\Public\Documents\ArcGIS\Default.gdb"
# Set output folder
output_folder = "C:\Users\Public\Documents\ArcGIS\Output"
# Set input files
origin_file = "C:\Users\Public\Documents\ArcGIS\Default.gdb\Origins"
destination_file = "C:\Users\Public\Documents\ArcGIS\Default.gdb\Destinations"
# Set analysis layer
analysis_layer = "C:\Users\Public\Documents\ArcGIS\Default.gdb\Network"
# Set parameters
parameters = {
    "ORIGIN": origin_file,
    "DESTINATION": destination_file,
    "ANALYSIS_LAYER": analysis_layer
}
# Run route tool
route_tool = "Route"
route_parameters = {
    "ORIGIN": parameters["ORIGIN"],
    "DESTINATION": parameters["DESTINATION"],
    "ANALYSIS_LAYER": parameters["ANALYSIS_LAYER"]
}
route_tool(arcpy, route_parameters)
# Create output shapefile
output_shapefile = os.path.join(output_folder, "Route.shp")
arcpy.CreateFeatureclass_management("Route", output_shapefile, "Line")
# Create output table
output_table = os.path.join(output_folder, "RouteBreakdown.csv")
arcpy.TableToDelimitedText_conversion("Route", output_table)
```

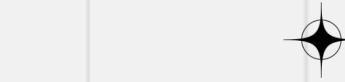
HOW THE TOOL LOOK LIKE IN GIS

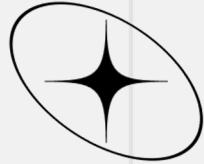




MUSA/CPLN CAPSTONE

# RESULTS & OUTPUT



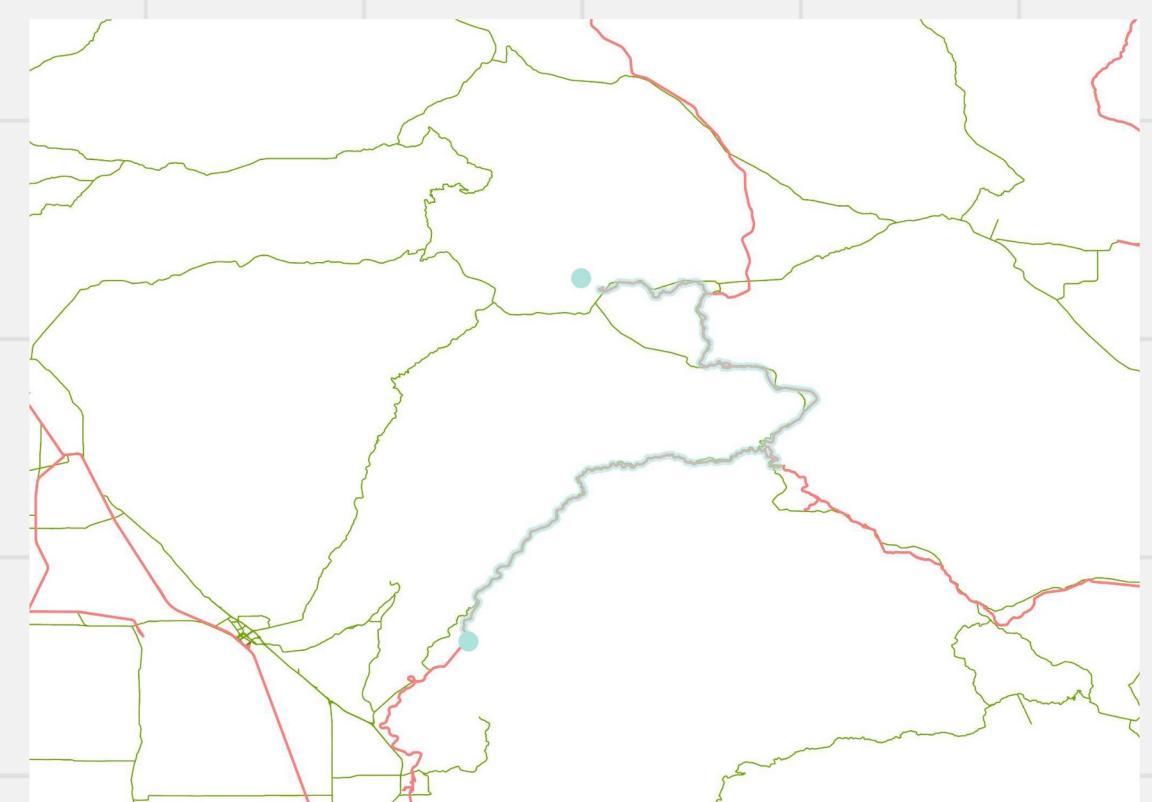
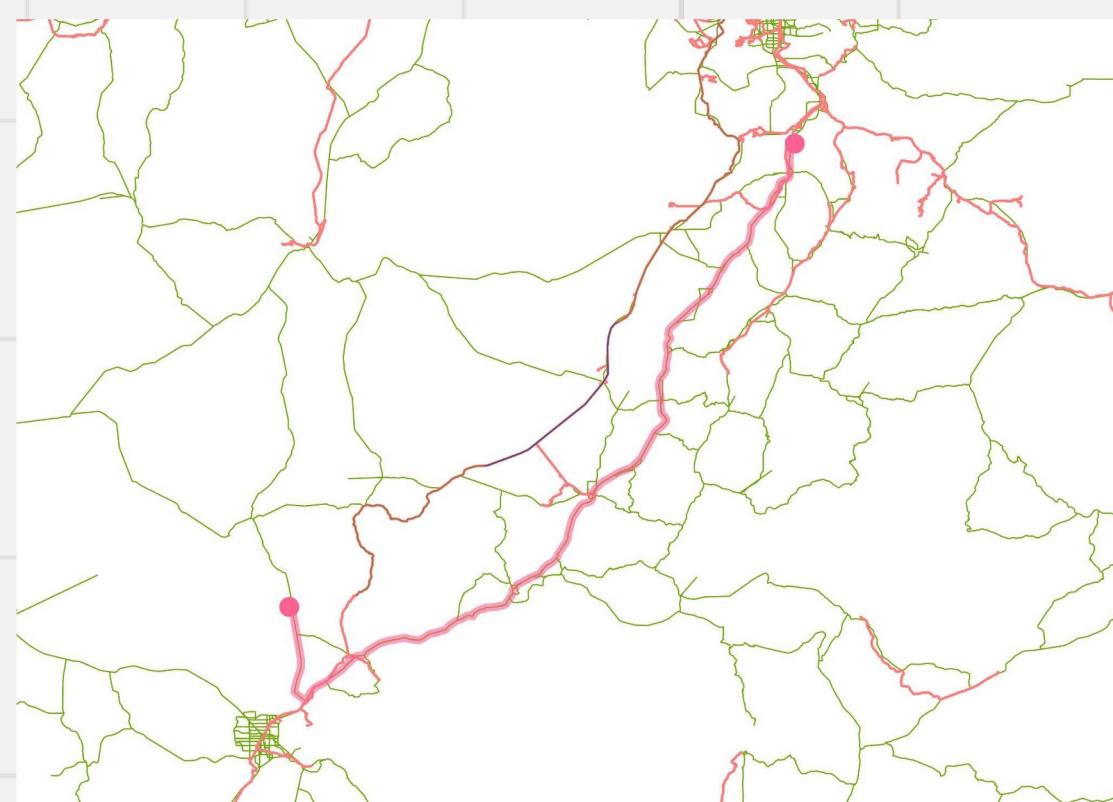
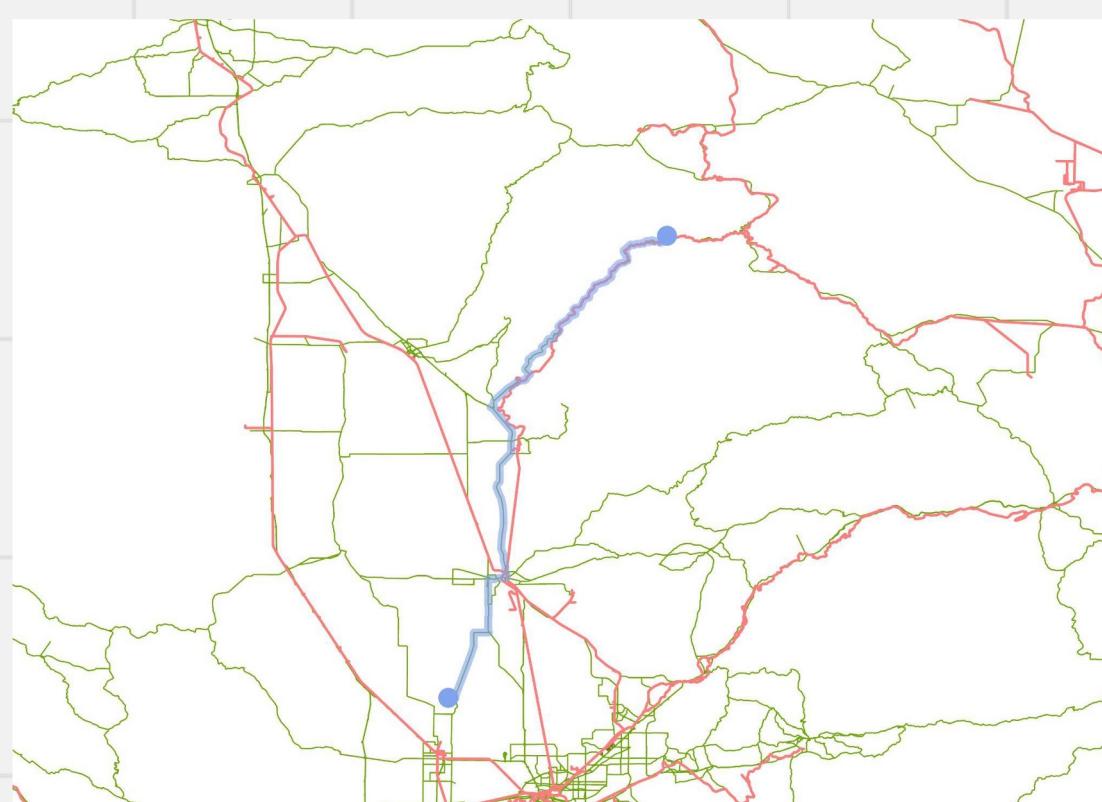


# OUTPUT

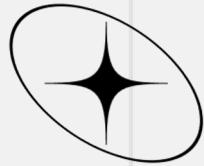
Routing	Origin-Destination Matrix
<ul style="list-style-type: none"><li>• Optimal Route for COST and RISK Impedance with...<ul style="list-style-type: none"><li>○ <math>\text{Truck}_{\text{Risk}}</math></li><li>○ <math>\text{Truck}_{\text{Cost}}</math></li><li>○ <math>\text{Rail}_{\text{Risk}}</math></li><li>○ <math>\text{Rail}_{\text{Cost}}</math></li><li>○ <math>\text{Pipeline}_{\text{Risk}}</math></li></ul></li></ul>	<ul style="list-style-type: none"><li>• Impedance values for solution pairs</li></ul>



# Route Shapefile



# Cost & Risk Impediment Breakdown



# Reference

NASA. (2024). The Keeling Curve. <https://climate.nasa.gov/vital-signs/carbon-dioxide/?intent=121>

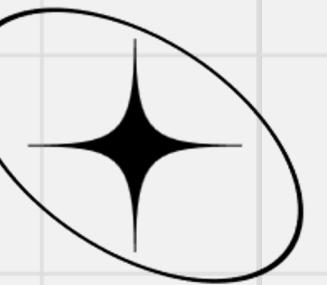
Roads to Removal. (2024). <https://roads2removal.org/>

Great Plains Institute. (2022). <https://betterenergy.org/blog/gpi-carbon-and-hydrogen-hubs-atlas/>

Statista. (2023). <https://www.statista.com/statistics/726634/large-scale-carbon-capture-and-storage-projects-worldwide-capacity/>

Congressional Budget Office. (2023). <https://www.cbo.gov/publication/59832>

IEA. (2024). <https://www.iea.org/energy-system/carbon-capture-utilisation-and-storage>



# THANK YOU

