NuLeptonSim: What Can It Do?

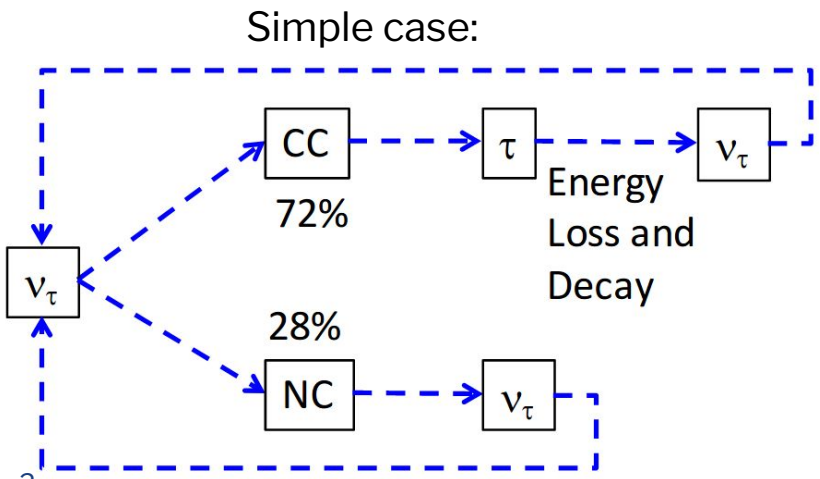
Austin Cummings

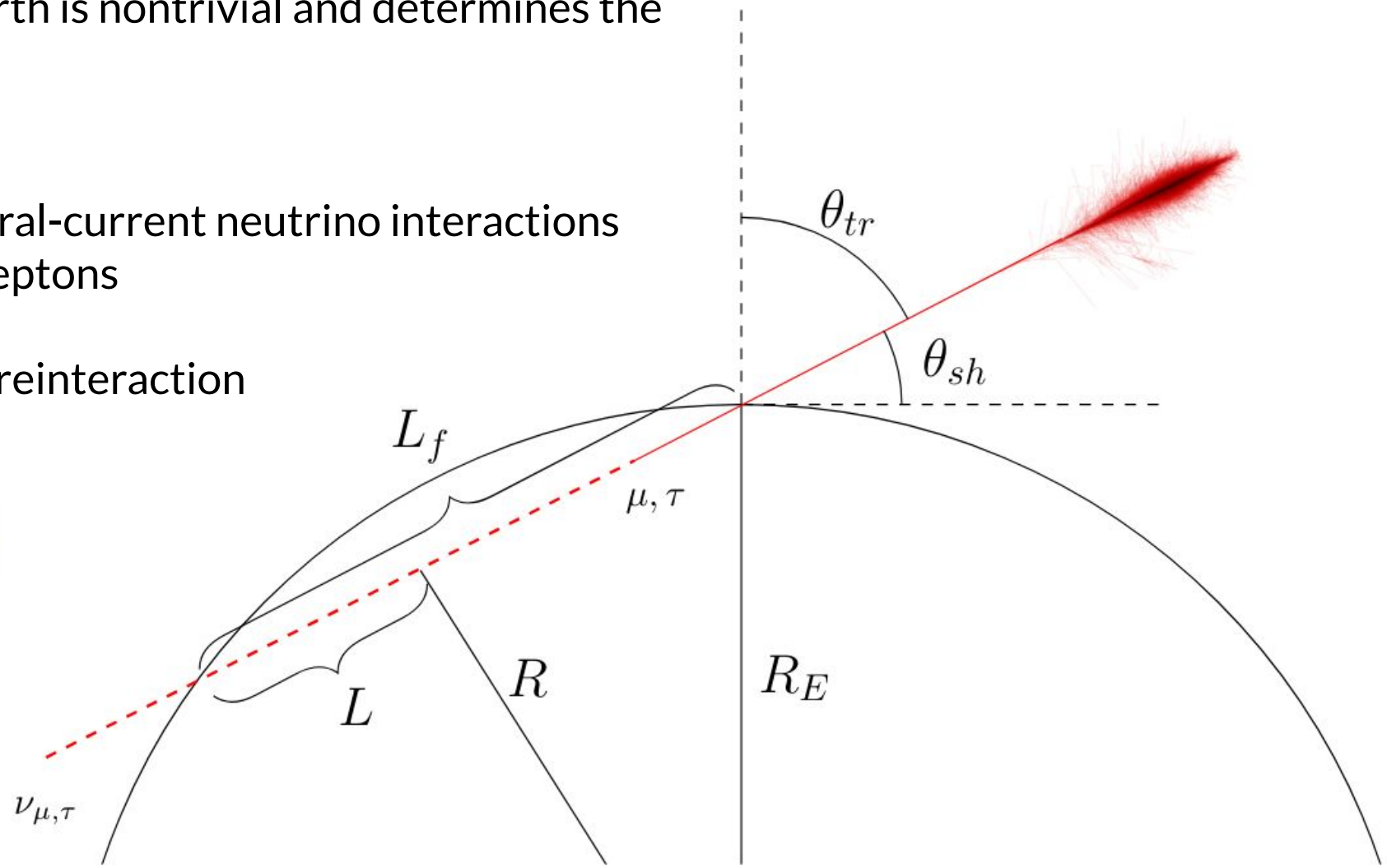
TAMBO-Trinity-BEACON/HERON Workshop, Harvard
10/31/2025

Modeling Neutrino Propagation

- ν_τ propagation through the Earth is nontrivial and determines the possible detection capabilities

- Must properly consider:
  - Charged-current and neutral-current neutrino interactions
  - Energy losses of charged leptons
  - Charged lepton decay
  - Multiple particle tracking/reinteraction

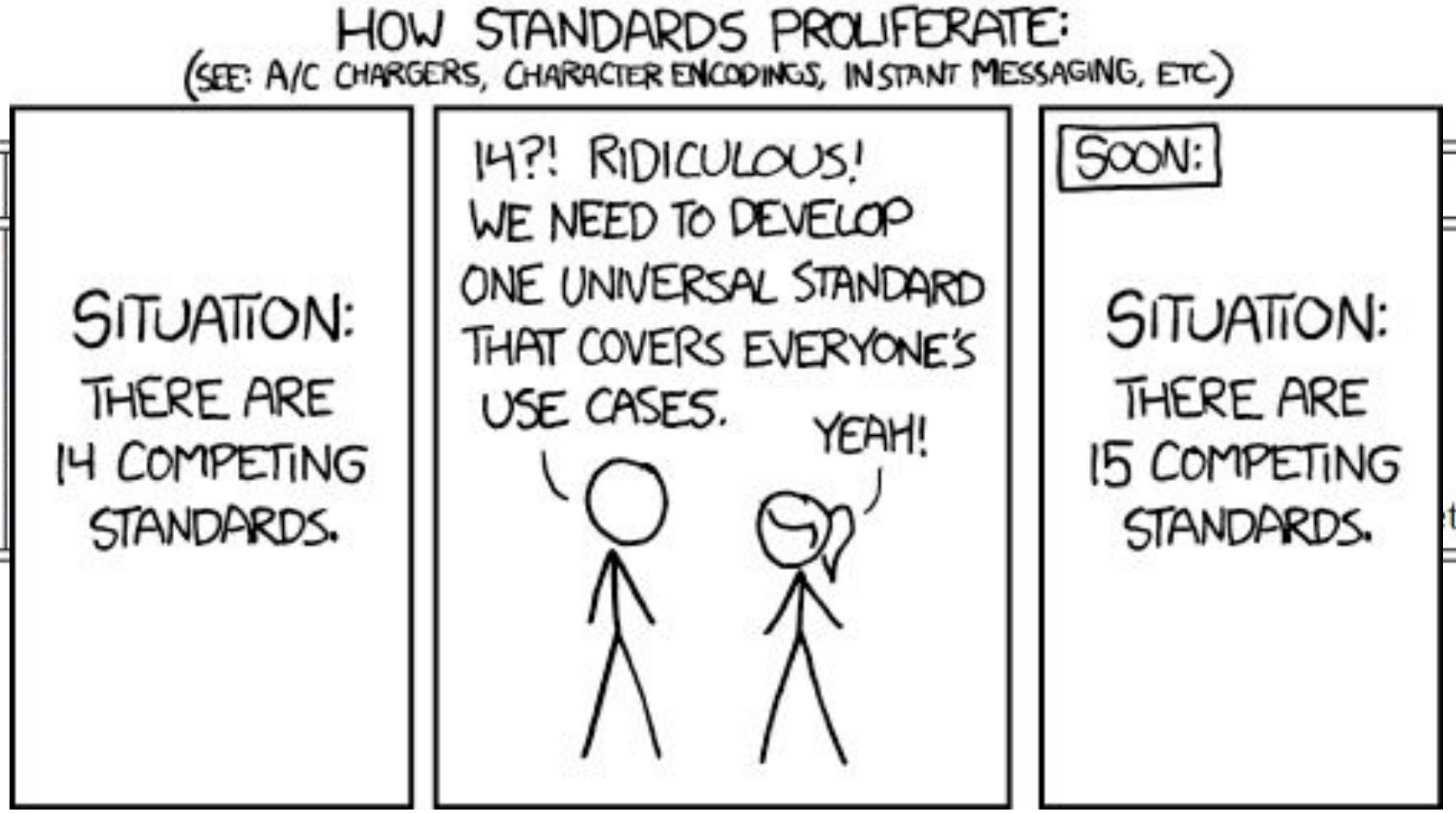




Existing Neutrino Propagation Tools

Software	Medium	Cross-Section	Energy Loss	Decay	Secondaries
`NuPropEarth`	PREM*	DIS+Others (GENIE)	PROPO./TAUSIC	TAUOLA	ν(all), τ
`TauRunner`	PREM, Sun*	DIS (Table)	PROPOSAL	Param.	ν(all), τ, μ
`nuPyProp`	PREM*	DIS (Table)	Table	Param.	τ
`NuTauSim`	PREM	DIS (Param.)	Param.**	Table	ν, τ
`Danton`	PREM*	DIS+GLRES (ENT)	PUMAS	TAUOLA (Alouette)	ν(all), τ

Existing Neutrino Propagation Tools



Software		Secondaries
NuPropEarth		ν(all), τ
TauRunner		ν(all), τ, μ
nuPyProp		τ
NuTauSim		ν, τ
Danton	tte)	ν(all), τ

What is NuLeptonSim?

- Based on the principles of NuTauSim:
  - Fast, while not sacrificing accuracy
  - Extremely modular, adaptable to different use cases

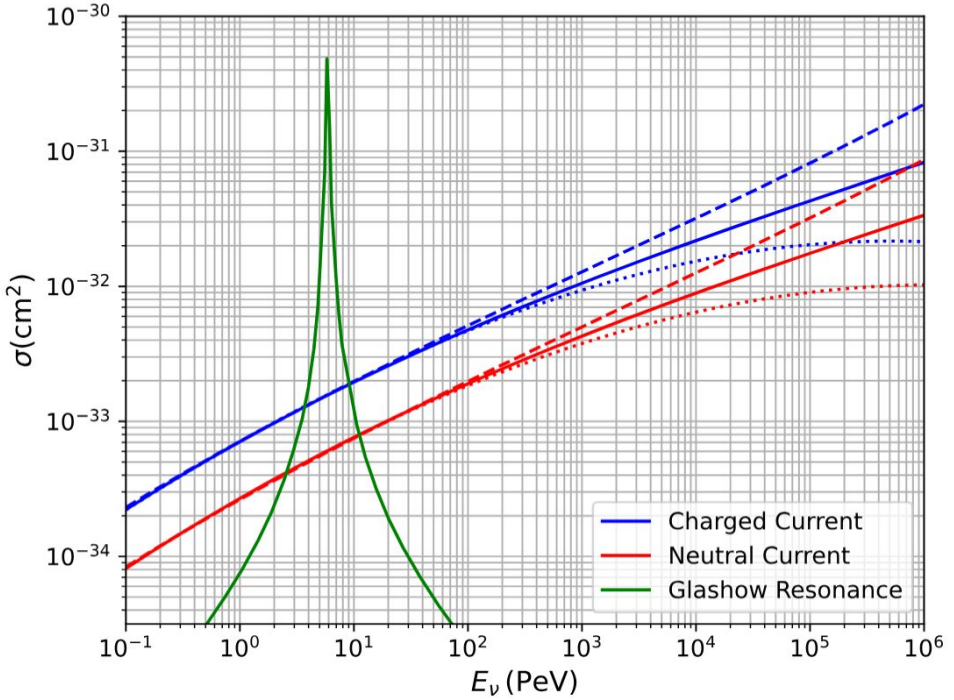
- Features of NuLeptonSim
  - Moves beyond just ν_τ, can also propagate ν_μ, and ν_e
  - Includes stochastic losses of charged leptons
  - Includes 3D propagation of particles and saving of in-matter interactions

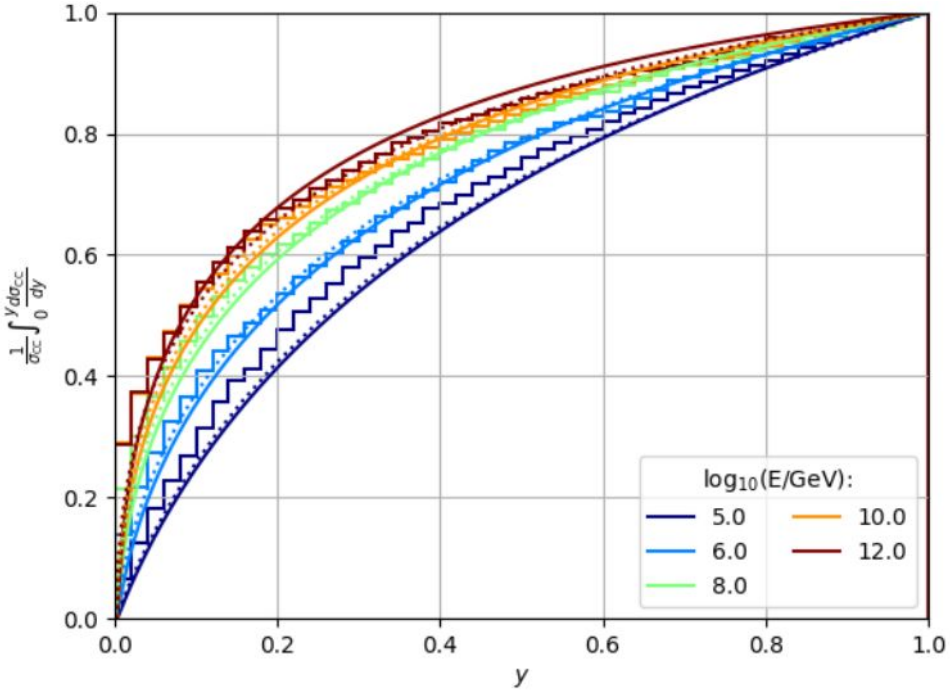
- Hopefully useful for a talk:
  - I will go through the different implementations of each of these in a bit of detail

Charged-current and neutral-current neutrino interactions
Energy losses of charged leptons
Charged lepton decay
Multiple particle tracking/reinteraction

Neutrino Cross Sections

- Extrapolations of charged and neutral current interactions from Phys. Rev. D 83, 113009
- Glashow resonance cross sections from Phys. Rev. D 98, 030001
- Inelasticity sampled from CTEQ5 tables
- Easy to swap with user-defined lookup tables
  - Recently, on my own branch implemented Phys. Rev. D 109, 113001
  - Can consider BSM scenarios





Charged Lepton Losses

- 2 loss model options:
  - Continuous

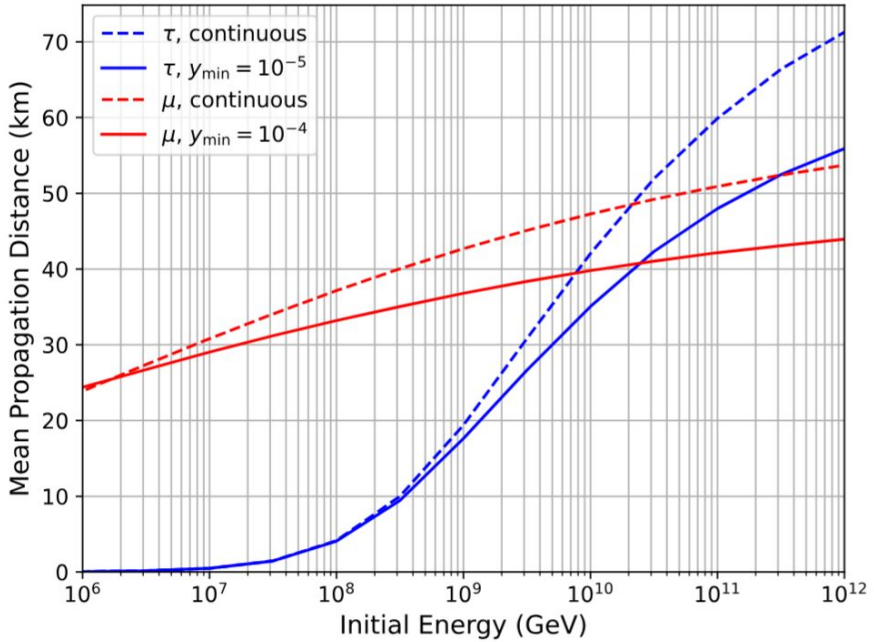
$$\left\langle \frac{dE}{dX} \right\rangle = -\alpha(E) - \beta(E)E; \quad \beta(E) = \sum_i \beta^i(E) = \frac{N}{A} \int_{y_{min}}^{y_{max}} y \frac{d\sigma^i(y,E)}{dy} dy$$

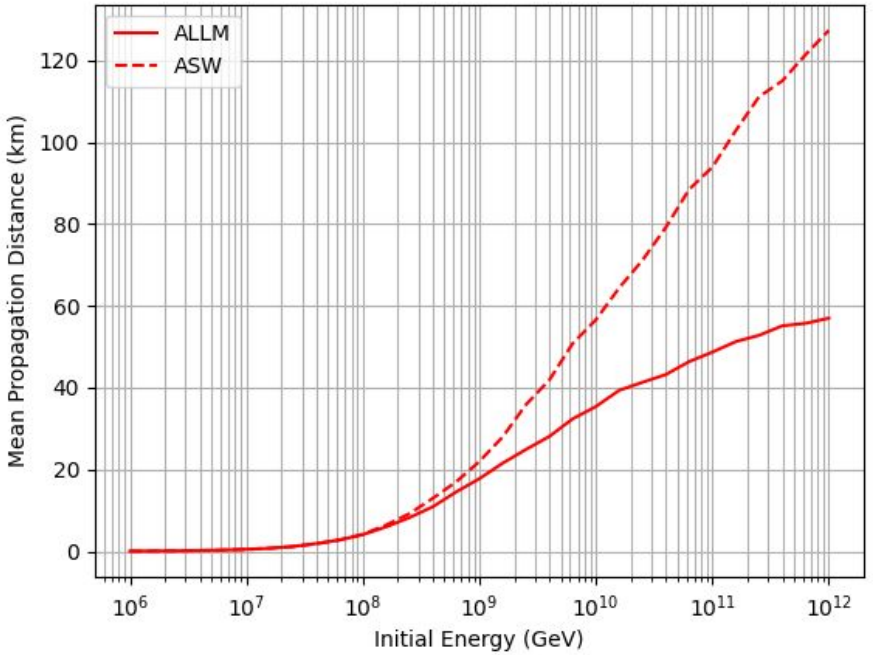
  - Stochastic

$$\mathrm{CDF}(y) = \int_{y_{min}}^{y} \frac{1}{\sigma_{tot}} \frac{d\sigma}{dy} dy$$

    and sample y randomly, above a value y_{min}
  - >10% on P_{exit} for $E_\nu > 10^{20}$eV
  - Required for in-matter interactions
  - Includes LPM suppression for muon bremsstrahlung

- User-swappable energy loss models
  - On my own branch, recently swapped stochastic ALLM with stochastic ASW





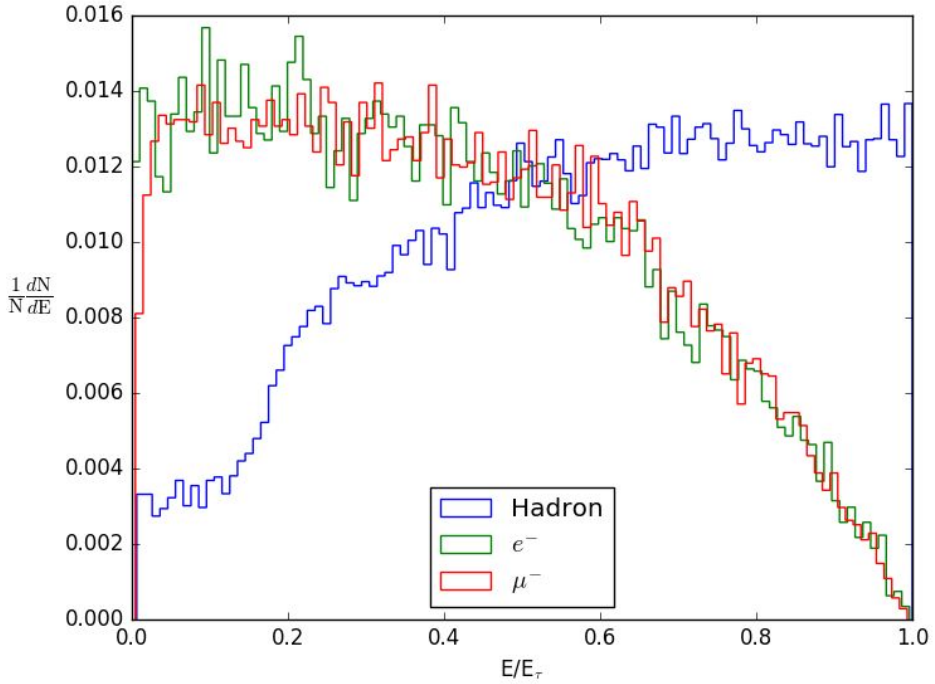
Decays

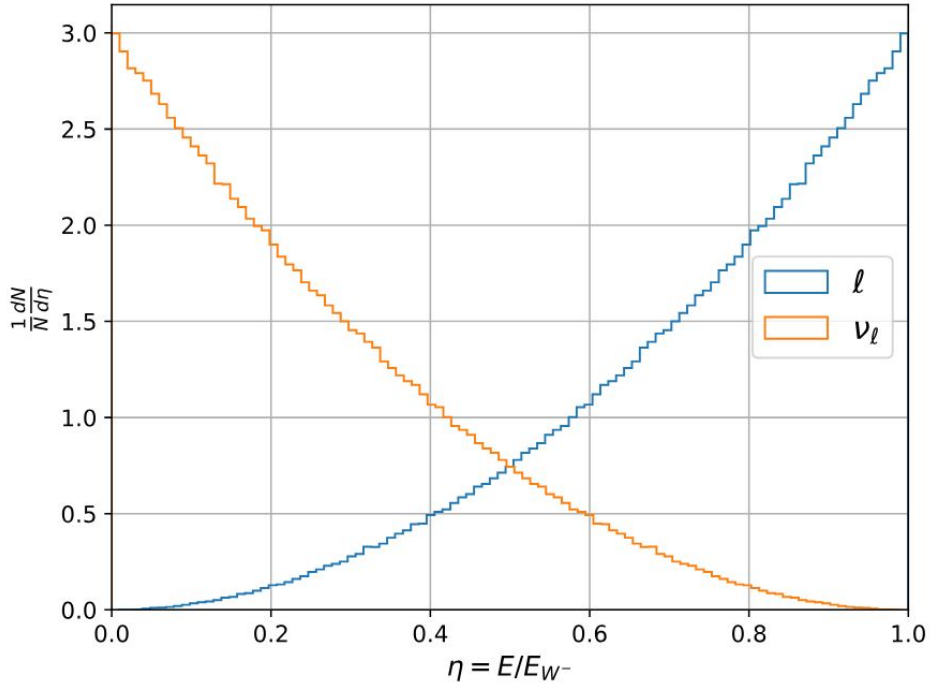
- Tau, muon decays sampled from Pythia tables
  - Assume $e^{\pm}$, hadrons interact immediately, all other leptons continue propagation
  - Ignore de-polarization effects, minimal effects from JCAP01(2023)041

- Kinematics of W- decay from Glashow resonance well known:

$$\frac{1}{N}\frac{dN}{d\cos\theta_l} = \frac{3}{16\pi}(1+\cos\theta_l)^2$$

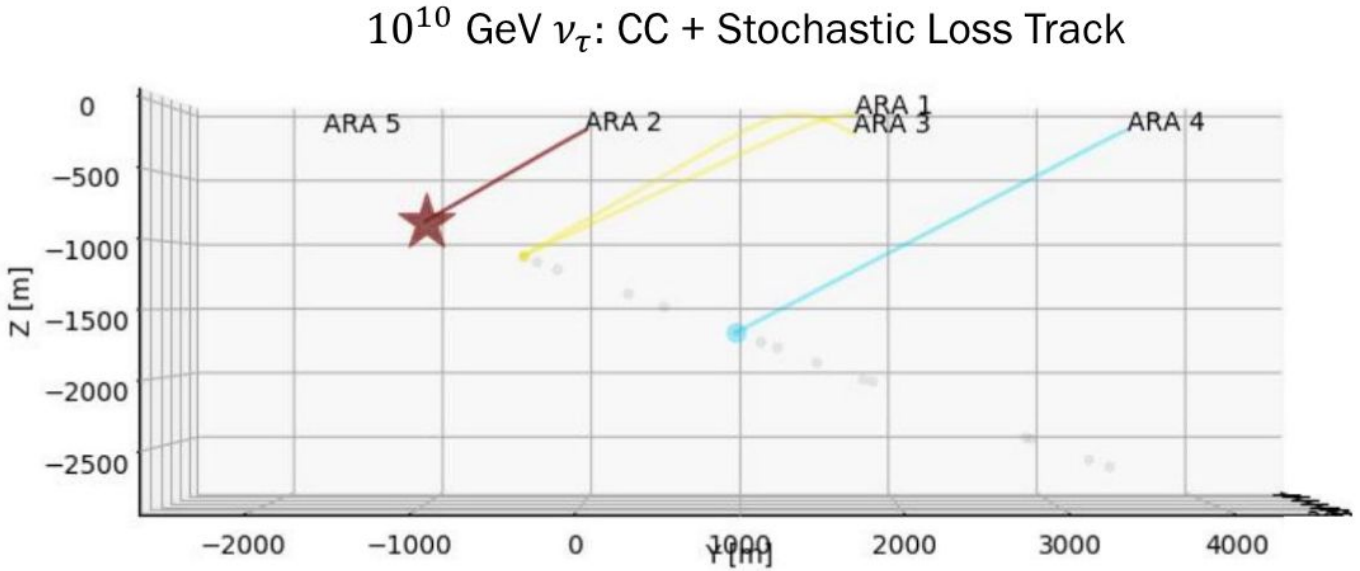
  Relativistic boost yields energy distribution



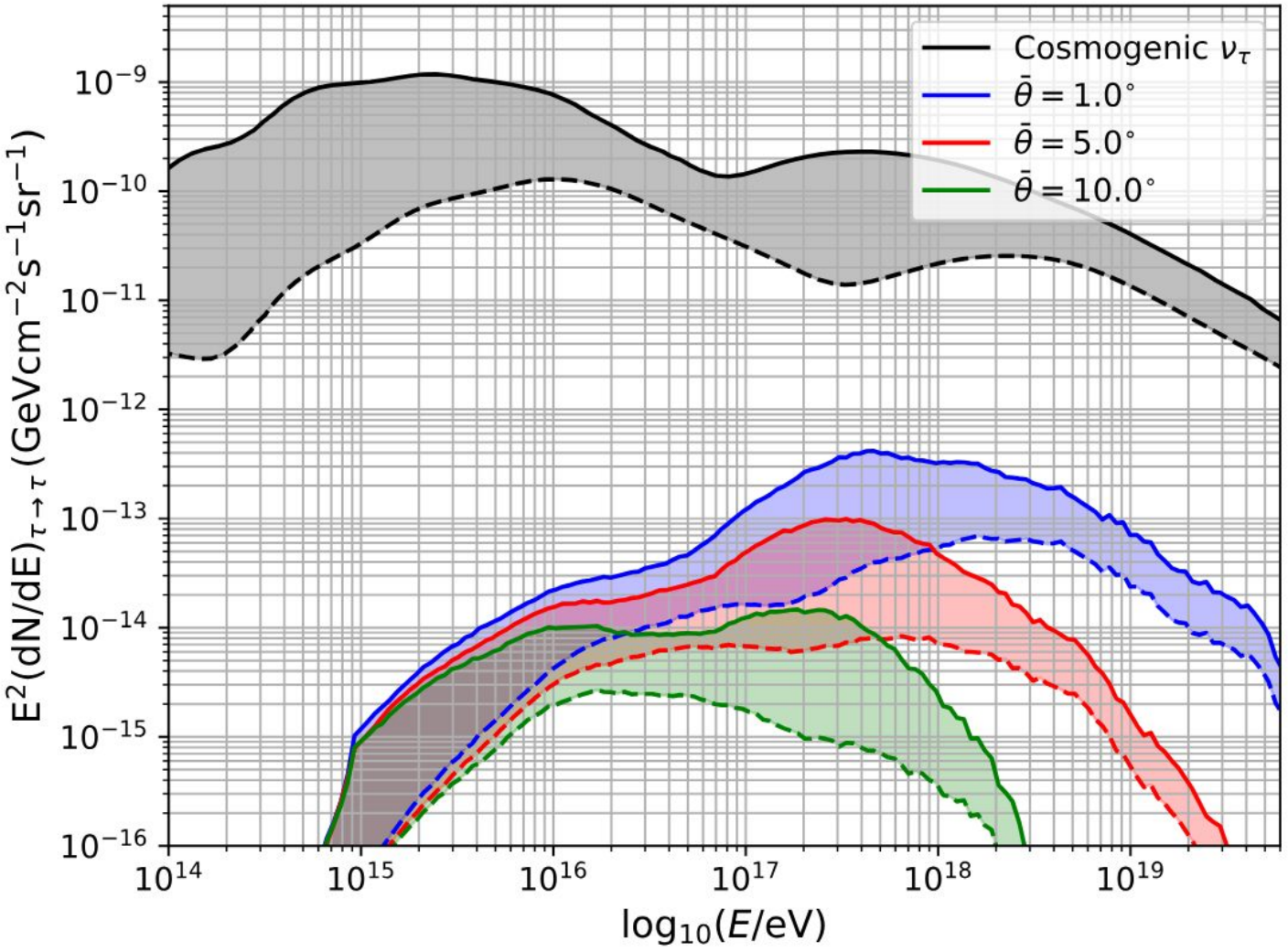


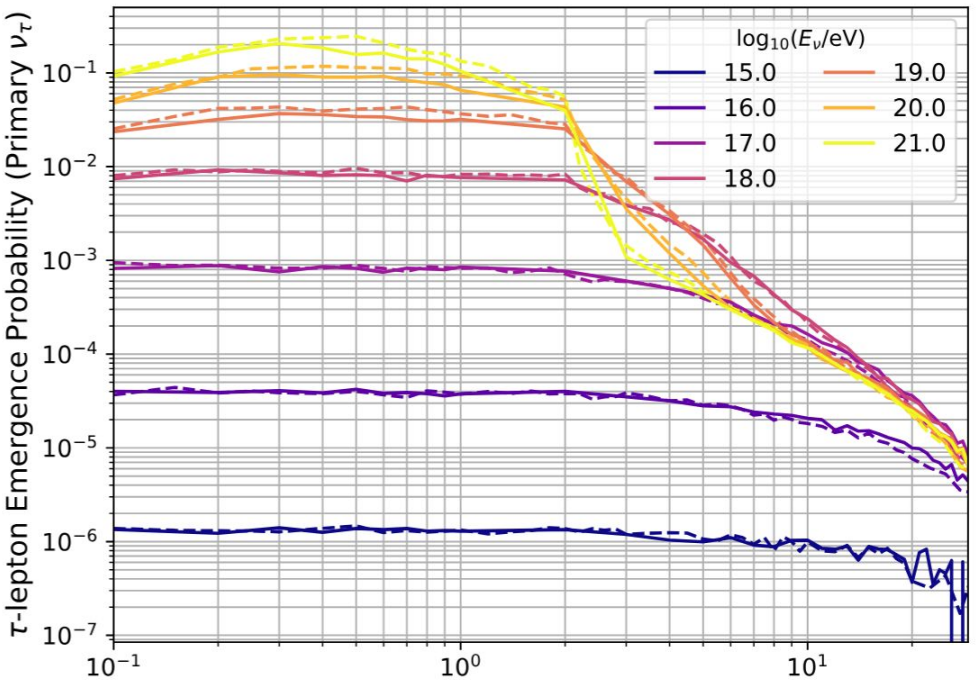
Multiple Particle Tracking/Saving Losses

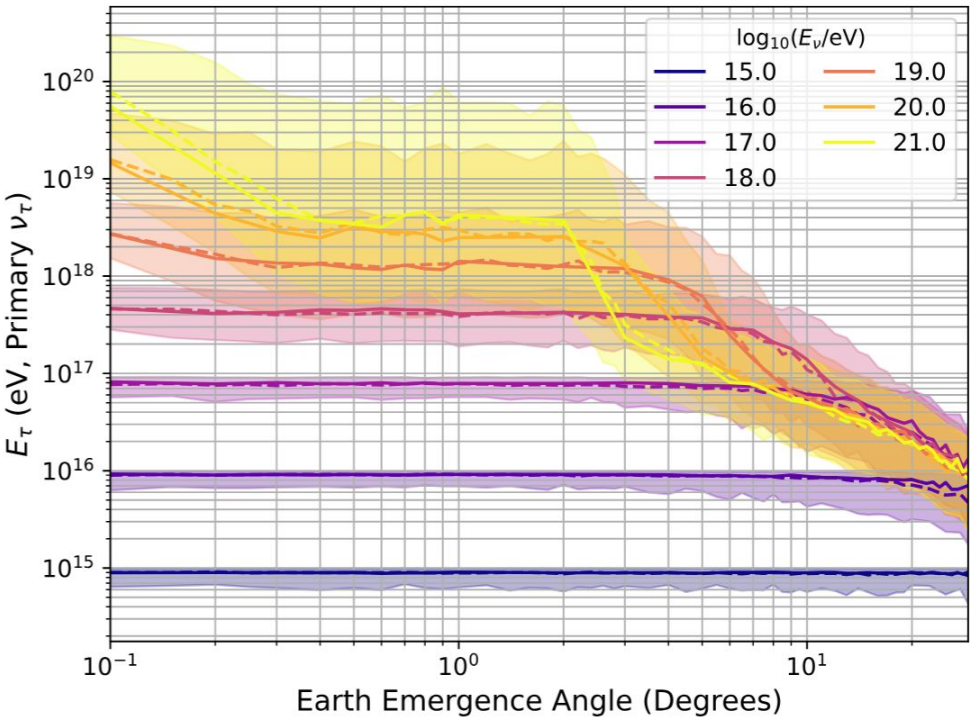
- Particle propagation now done in 3D
  - Particles that can continue propagation (all leptons but electrons) are kept in a stack
- Can define a volume within which interactions above a user defined energy are kept (energy, interaction type, direction)--used for ARA, PUEO, RET
  - Sphere
  - Cylinder
  - Cube
- In principle, this could also be used with a topographic simulation to define X(L), though generating a lookup table would be nontrivial



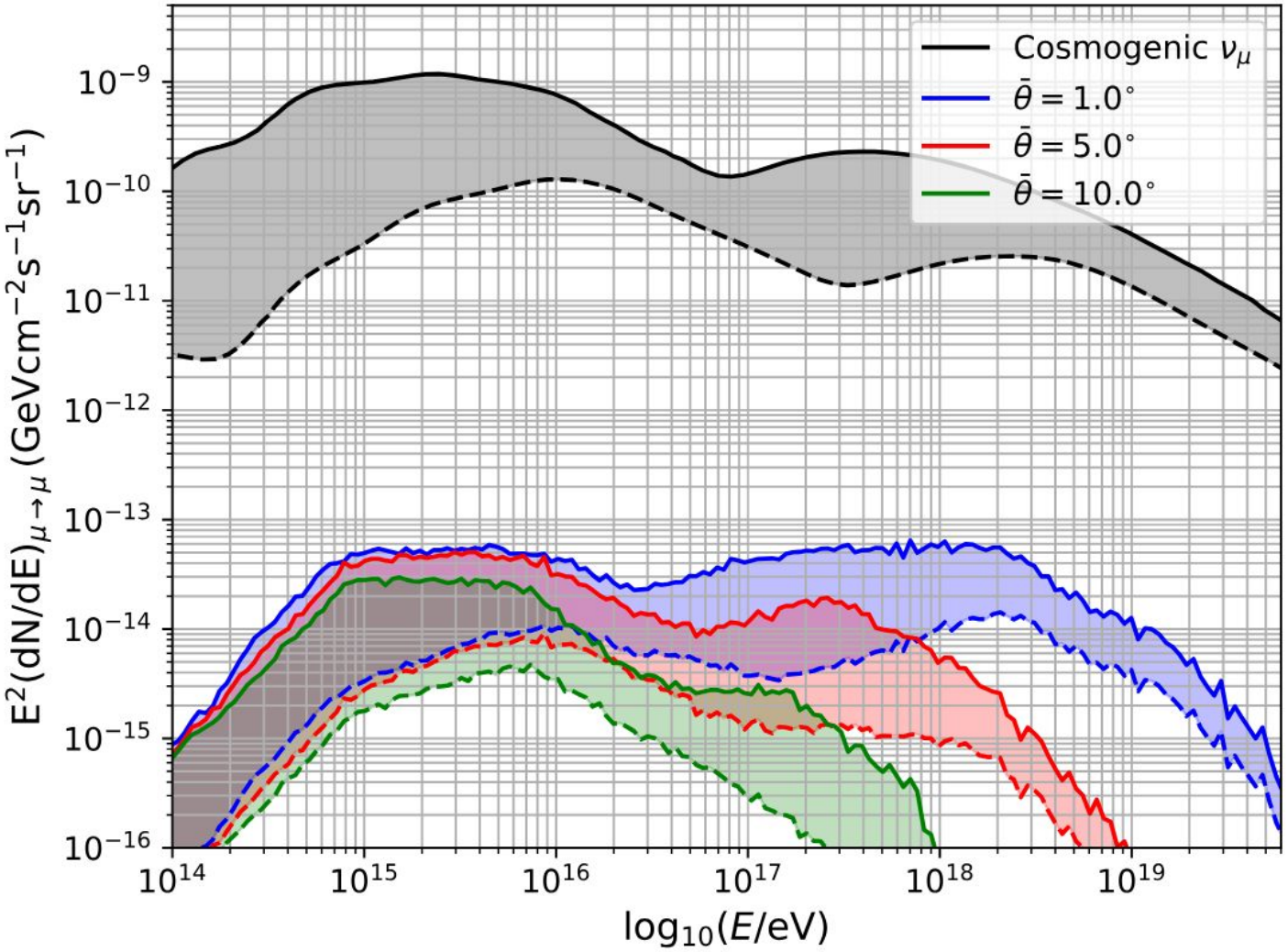
Some Results (taus)

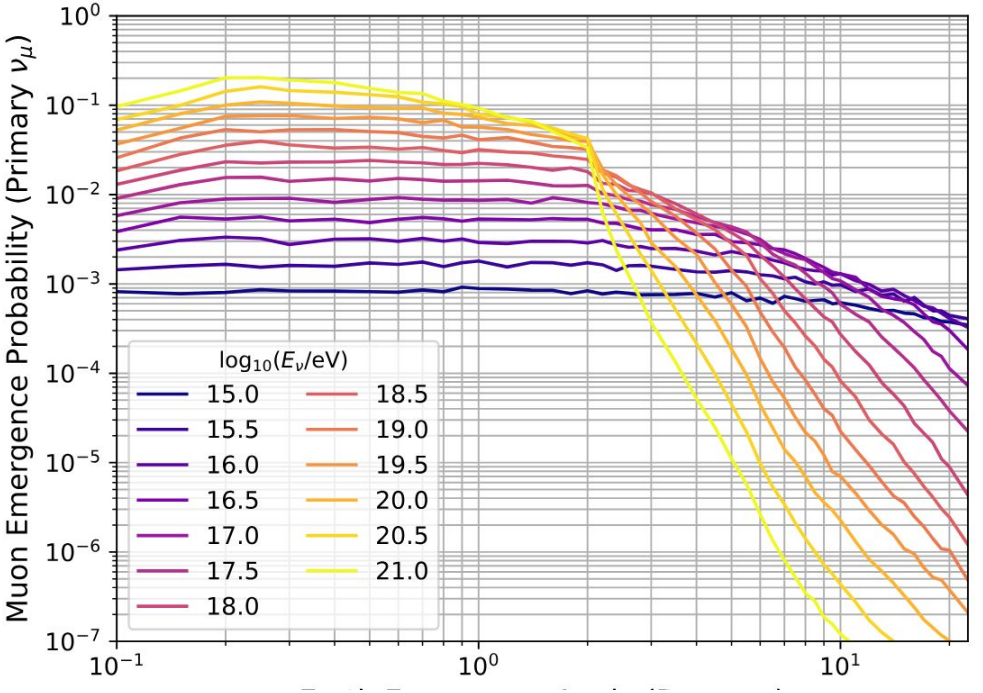


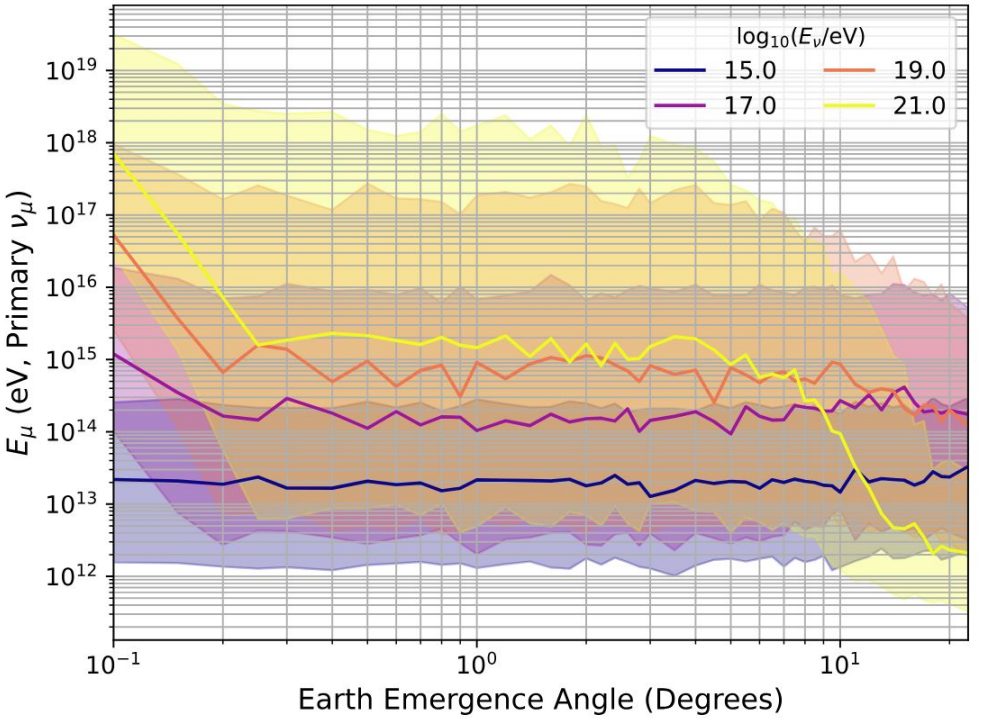




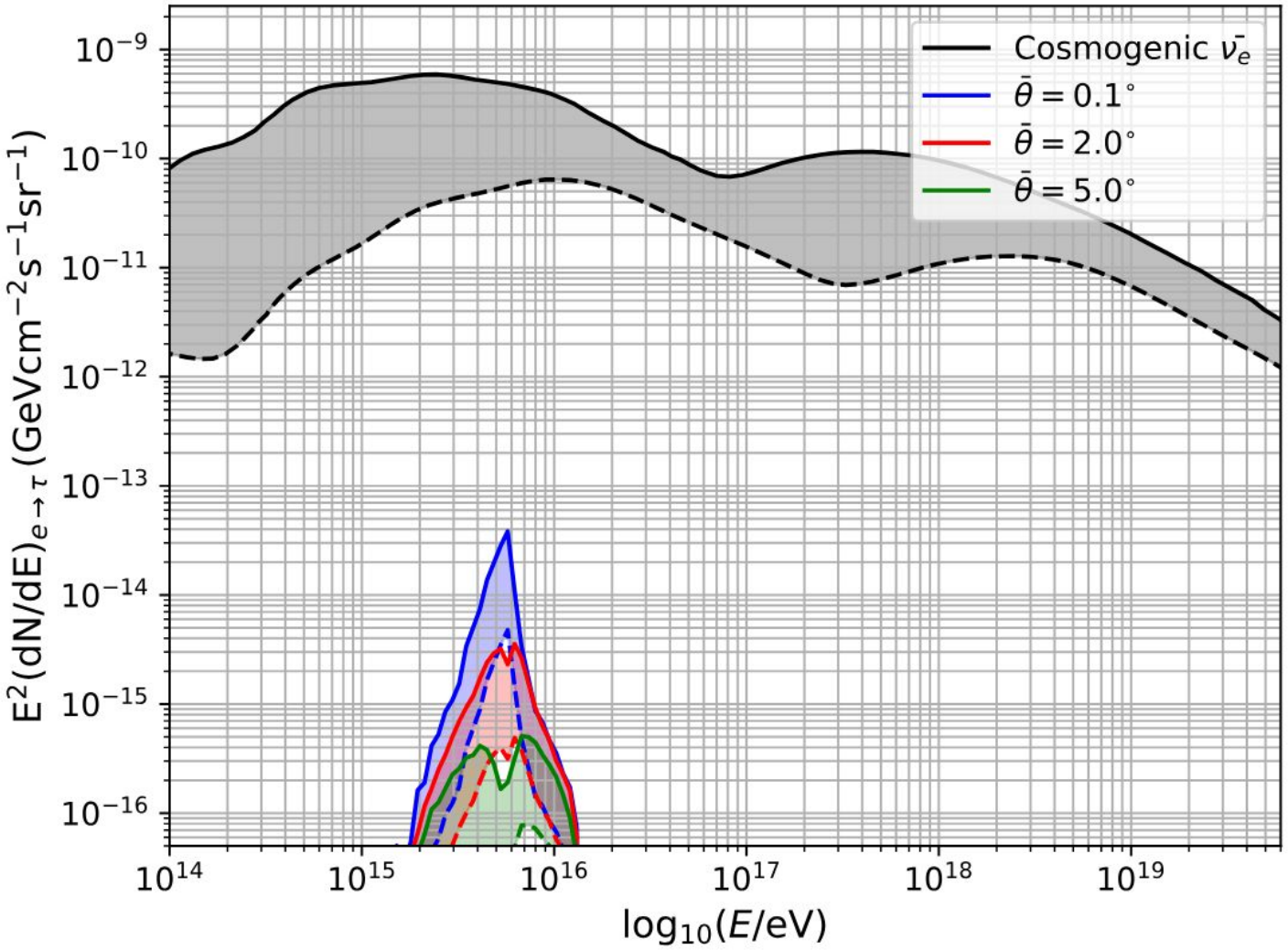
Some Results (muons)

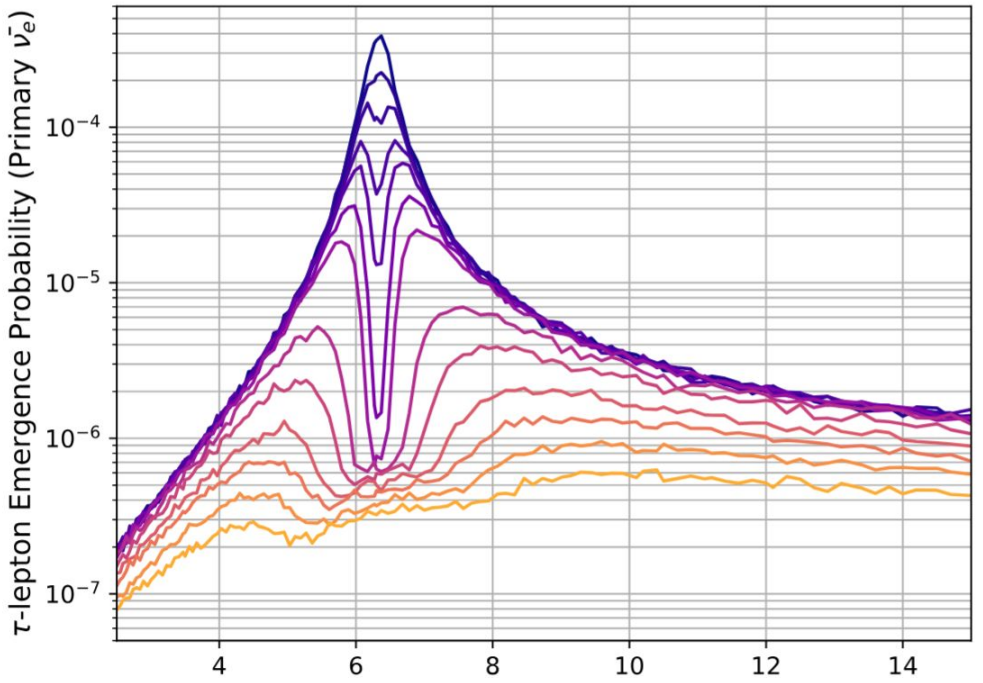


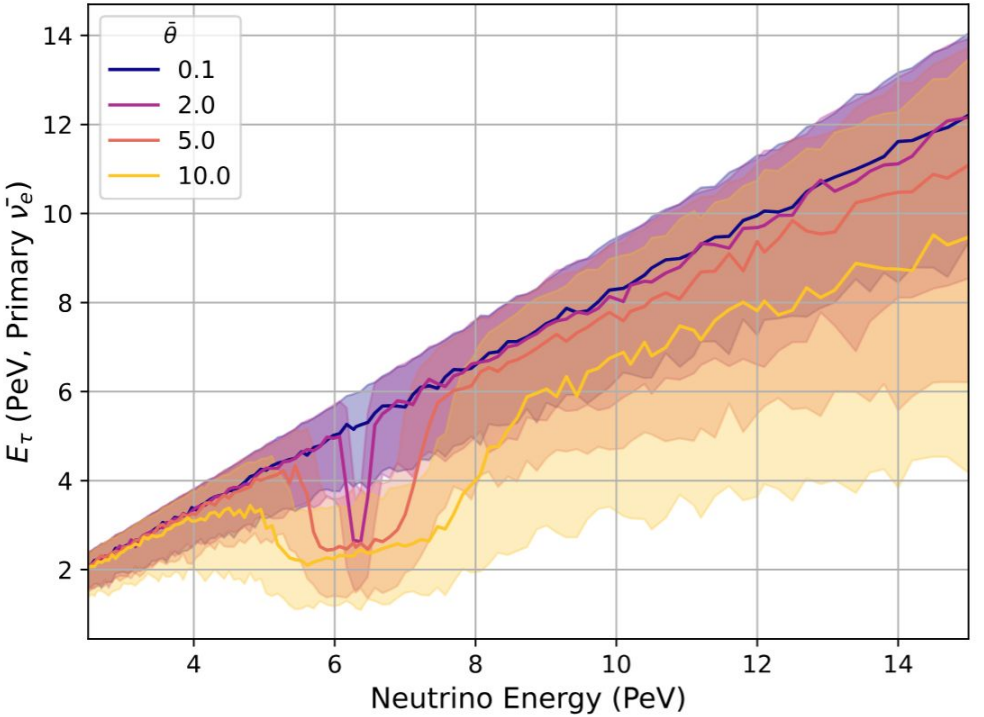




Some Results (Glashow)







```
#########################################################################################################################

# loading data from NuTauSim (cite https://arxiv.org/abs/1901.08498) and building interpolators

# the probability interpolator takes as arguments N pairs of (log10(neutrino_energy(eV)), Earth emergence angle (degrees))
# and returns an array of length N representing the log of the Earth emergence probability for each pair
prob_data = np.load("Tau_Emergence_Probabilities.npy", allow_pickle = True)
energies, angles, probabilities = np.log10(prob_data[0]), prob_data[1], np.log10(prob_data[2])
prob_interpolator = RegularGridInterpolator((energies, angles), probabilities, bounds_error=False, fill_value = -10)

# the energy interpolator takes as arguments N pairs of (log10(neutrino_energy(eV)), Earth emergence angle (degrees), y)
# and returns an array of length N representing the log of a randomly sampled tau-lepton fractional energy (Etau/Eneutrino) for each triple
# the actual random sampling (and need for the value y, which is randomly sampled) is done via inverse tranform sampling: https://en.wikipedia.org/wiki/Inverse_transform_sampling
cdf_data = np.load("Tau_Emergence_Energy_CDFs.npy", allow_pickle = True)
energies, angles, cdfs, cdfxs = np.log10(cdf_data[0]), cdf_data[1], cdf_data[2], cdf_data[3]
energy_interpolator = RegularGridInterpolator((energies, angles, cdfs), cdfxs, bounds_error=False, fill_value = -10)

# one note of caution: my data was not generated below Earth emergence angles of 0.1 degrees, so results below that are undefined

#########################################################################################################################

# example use 1: generate an Earth emergence probability curve

N = 1000 # number of angular samples
angles = np.linspace(0.1,30,N) # spacing of Earth emergence angle (in degrees)
ens = 1e18*np.ones(N) # input neutrino energy in eV (in this case, the same for every angle, but doesn't need to be)
probs = 10**prob_interpolator(np.array([np.log10(ens), angles]).T) # probability of Earth emergence

fig = plt.figure()
plt.plot(angles, probs)
plt.xscale('log')
plt.yscale('log')
plt.xlabel('Earth Emergence Angle (Degrees)')
plt.ylabel('Earth Emergence Probability (Pexit)')
plt.grid(which = 'both')

#########################################################################################################################

# example use 2: get emergent energies given input events (how you will likely use this in your MC)

N = 1000000 # number of taus you want to simulate (may need to do this in a for loop if RAM has too high of a usage)
angles = 30*np.random.rand(N) # Earth emergence angle of each event (in degrees). Currently, uniformly random. This will be replaced by your MC
ens = 1e17*np.ones(N) # input neutrino energy in eV (in this case, the same for every angle, but doesn't need to be)
sample_ys = np.random.rand(N) # a random value between 0 and 1 to do inverse transform sampling on the tau energy CDF tables

# my data only goes down to 0.1 degree Earth emergence angle, so below that, results are undefined and you need to be somewhat careful
valid_indices = angles >= 0.1
angles = angles[valid_indices]
ens = ens[valid_indices]
sample_ys = sample_ys[valid_indices]

energies = ens*10**energy_interpolator(np.array([np.log10(ens), angles, sample_ys]).T) # energies of Earth emergent tau leptons

fig = plt.figure()
plt.hist(np.log10(energies), bins = 50, density = True)
plt.xlabel(r'$E_{\tau}$'+' (eV)')
plt.show()

#########################################################################################################################
```

```
##########################################################################################################

# loading data from NuTauSim (cite https://arxiv.org/abs/1901.08498) and building interpolators

# the probability interpolator takes as arguments N pairs of (log10(neutrino_energy(eV)), Earth emergence angle (degrees))
# and returns an array of length N representing the log of the Earth emergence probability for each pair
prob_data = np.load("Tau_Emergence_Probabilities.npy", allow_pickle = True)
energies, angles, probabilities = np.log10(prob_data[0]), prob_data[1], np.log10(prob_data[2])
prob_interpolator = RegularGridInterpolator((energies, angles), probabilities, bounds_error=False, fill_value = -10)

# the energy interpolator takes as arguments N pairs of (log10(neutrino_energy(eV)), Earth emergence angle (degrees), y)
# and returns an array of length N representing the log of a randomly sampled tau-lepton fractional energy (Etau/Eneutrino) for each triple
# the actual random sampling (and need for the value y, which is randomly sampled) is done via inverse tranform sampling: https://en.wikipedia.org/wiki/Inverse_transform_sampling
cdf_data = np.load("Tau_Emergence_Energy_CDFs.npy", allow_pickle = True)
energies, angles, cdfs, cdfxs = np.log10(cdf_data[0]), cdf_data[1], cdf_data[2], cdf_data[3]
energy_interpolator = RegularGridInterpolator((energies, angles, cdfs), cdfxs, bounds_error=False, fill_value = -10)

# one note of c                                                                    es, so results

##########                                                                        ##########

# example use 1

N = 1000 # numb
angles = np.lin
ens = 1e18*np.c                                                                  e, but doesn't
probs = 10**pro                                                                  emergence

fig = plt.figur
plt.plot(angles
plt.xscale('log
plt.yscale('log
plt.xlabel('Ear
plt.ylabel('Ear
plt.grid(which

##########                                                                        ##########

# example use 2                                                                   your MC)

N = 1000000 # n                                                                  M has too high
angles = 30*np.                                                                  y, uniformly ra
ens = 1e17*np.c                                                                  e, but doesn't
sample_ys = np.                                                                  oling on the ta

# my data only goes down to 0.1 degree Earth emergence angle, so below that, results are undefined and you need to be somewhat careful
valid_indices = angles >= 0.1
angles = angles[valid_indices]
ens = ens[valid_indices]
sample_ys = sample_ys[valid_indices]

energies = ens*10**energy_interpolator(np.array([np.log10(ens), angles, sample_ys]).T) # energies of Earth emergent tau leptons

fig = plt.figure()
plt.hist(np.log10(energies), bins = 50, density = True)
plt.xlabel(r'$E_{\tau}$'+' (eV)')
plt.show()

##########################################################################################################
```