فهرست

فهرست 2
مقدمه 3
توابع استفاده شده برای پیاده سازی الگوریتم ها 4
الگوریتم MACTER 5
الگوریتم CTORA 5
الگوریتم CODO 5
مرحله همکاری Collaborative Step 5
مرحله تصمیم گیری انتقال کار Offloading Decision 6
تکرار مراحل: 6
تابع هیوریستیک heuristic_scheme 6
محاسبه زمان محلی local_time 7
محاسبه مصرف انرژی محلی local_energy 7
محاسبه زمان و انرژی برای انتقال به سرور VEC_time و VEC_energy 7
تصمیم‌گیری 7
تحلیل نتایج 7
کاربرد در شرایط خاص 12

مقدمه

در دنیای امروز، با پیشرفت فناوری و گسترش استفاده از خودروهای الکتریکی و شبکه‌های ارتباطی، نیاز به بهبود بهره‌وری محاسباتی در سیستم‌های کامپیوتری مرتبط با این خودروها بیش از پیش احساس می‌شود. یکی از چالش‌های اساسی در این زمینه، تخصیص بهینه منابع و انتقال وظایف محاسباتی (Offloading Task) به بخش‌های مختلف شبکه است. این چالش در معماری سه‌لایه شامل خودروهای الکتریکی، ایستگاه‌های پایه و سرورهای ابری نمود بیشتری پیدا می‌کند.

هدف اصلی این پروژه، افزایش بهره‌وری محاسباتی شبکه‌های ارتباطی خودروهای الکتریکی از طریق بهینه‌سازی استراتژی‌های Offloading Task و تخصیص منابع است. بهره‌وری محاسباتی به عنوان نسبت بیت‌های محاسبه شده به کل انرژی مصرف شده در خودروهای الکتریکی تعریف می‌شود. در این راستا، با تعریف یک مسئله بهینه‌سازی و استفاده از نظریه بازی‌ها، تلاش می‌شود تا بهره‌وری سیستم به حداکثر برسد.

روش پیشنهادی در این پروژه شامل استفاده از تکرارهای وراثتی و رویکردهای بازی‌محور برای اتخاذ سیاست‌های Offloading Task و تخصیص منابع است. این رویکرد به دنبال یافتن بهترین جواب ممکن با توجه به شرایط و محدودیت‌های موجود است. هدف از پیاده‌سازی این روش، مقایسه نتایج حاصل با روش‌های پایه موجود در این حوزه، نظیر MACTER و CTORA، و بررسی میزان بهبود حاصل از به کارگیری روش پیشنهادی است.

توابع استفاده شده برای پیاده سازی الگوریتم ها

به ازای هر یک از حالات اجرای تسک local و استفاده از VEC، توابع مربوط به محاسبه زمان اجرا، انرژی مصرفی و utility نوشته شده است:

- Local_computation_time: زمان اجرا در حالت local
- VEC_computation_time: زمان اجرا در حالت VEC
- Local_enegy_consumption: انرژی مصرفی در حالت local
- VEC_energy_consumption: انرژی مصرفی در حالت VEC
- Utility_VEC: مقدار utility در حالت VEC
- Utility_local: مقدار utility در حالت local

در ابتدا مقادیر تعدادی از متغیر های اولیع مقداردهی می‌شوند مانند task_data_size که حجم داده ی هر تسک را نشان می‌دهد.
vehicle_computation_capacity که ظرفیت محاسباتی هر وسیله را نشان می‌دهد. دهد. Position که مکان هر وسیله در آن لحظه را نشان می‌دهد و ...

```
task_data_size = np.random.uniform(150, 300, num_vehicles)
vehicle_computation_capacity = np.random.uniform(0.5, 2, num_vehicles)
service_coefficient = vehicle_computation_capacity / task_data_size
position = [i * 10 for i in range(num_vehicles)]
transition_power = [random.randint(30, 50) for i in range(num_vehicles)]
distance_traveled = [(r * math.ceil(position[i] / 3)) - position[i] for i in range(num_vehicles)]
transmission_rate = []
for i in range(num_vehicles):
    if distance_traveled[i] != 0:
        a = math.pow(distance_traveled[i], -gamma)
    else:
        a = 1
    b = ((transition_power[i] * a * uplink_channel) / sigma_uu)
    transmission_rate.append(w_uu * math.log2(1 + b))

VEC_computation_capacity = np.random.uniform(20, 30)
network_latency = np.random.uniform(0.1, 0.5, num_vehicles)
t_max = np.random.uniform(10, 20, num_vehicles)
speed = np.random.uniform(30, 60, num_vehicles)
t_stay = [2 * (math.sqrt((r * r) - (e * e))) / speed[i] for i in range(num_vehicles)]
t_ptd = [min(t_stay[i], t_max[i]) for i in range(num_vehicles)]
```

سپس الگوریتم های مورد نظر روی این وسیله ها و متغیر های مربوط به آن‌ها اجرا می‌شود

الگوریتم MACTER

ابتدا یک decision strategy اولیه که در این کد به صورت آرایه ای از صفر در نظر گرفته شده تعیین می‌شود (یعنی هیچ تسکی offload نمی‌شود). سپس به تعداد iteration های مشخص شده تلاش می‌کنیم decision strategy را بهتر کنیم.

به این صورت که ابتدا یک تابع که مقدار objective را محاسبه می‌کند تعریف کرده که این تابع مجموع utility ها را خروجی می‌دهد که در حالتی که تسک offload شده باشد utility_vec و در غیر اینصورت باید utility_loc را به مجموع اضافه کند.

سپس روش بهینه resource allocation از روی objective با Resource ها محاسبه می‌شود.

در قدم دوم الگوریتم هم، به ازای شرایط جدید Resource allocation و خروجی مراحل قبل، و با محاسبه مجدد utility_loc و utility_vec و مقایسه آن‌ها، استراتژی offloading جدید به ازای هر وسیله تعیین می‌شود به این صورت که اگر utility_vec بیشتر باشد d[i] برابر با 1 و در غیر اینصورت برابر با 0 خواهد بود.

در انتها و پس از گذشت چند iteration جواب به مقدار تعادل نش نزدیک خواهد شد یعنی هیچ وسیله ای با تغییر استراتژی خود نمی‌تواند utility را بهتر کند.

الگوریتم CTORA

این الگوریتم هم مشابه الگوریتم MACTER عمل می‌کند با این تفاوت که iteration ندارد و به صورت مستقیم و بدون تکرار، توسط محاسبه utility_vec و utility_loc و مقایسه آن‌ها، offloading strategy برای هر وسیله تعیین می‌شود.

الگوریتم CODO [1]

به منظور بهینه‌سازی تصمیمات مرتبط با انتقال کار از خودروها به سرورهای VEC در شبکه‌های وسایل نقلیه استفاده می‌شود. این الگوریتم به صورت زیر عمل می‌کند:

مرحله همکاری (Collaborative Step):

در این مرحله، منابع سرورهای VEC بر اساس تصمیمات انتقال کار (offloading decisions) تنظیم می‌شوند.

---

[1] Collaborative Offloading Decision Optimization

برای هر خودرو، اگر تصمیم انتقال کار به VEC اتخاذ شده باشد، ابتدا محاسبه می‌شود که آیا افزایش یا کاهش منابع VEC مناسب است. این تصمیم بر اساس اختلاف میان میزان کارایی محاسباتی که VEC می‌تواند ارائه دهد و میانگین منابع فعلی VEC انجام می‌شود.
اگر کارایی محاسباتی VEC بیشتر از میانگین منابع فعلی VEC باشد، منابع VEC افزایش می‌یابند تا به حداکثر برسند. در غیر این صورت، منابع کاهش می‌یابند.

مرحله تصمیم گیری انتقال کار (Offloading Decision):

بعد از تنظیم منابع VEC در مرحله همکاری، هر خودرو تصمیم می‌گیرد که آیا کار را محاسبه محلی انجام دهد یا به VEC منتقل کند.
برای هر خودرو، محاسبه می‌شود که آیا کارایی محاسباتی VEC بیشتر از محاسبه محلی است یا خیر. اگر بله، تصمیم انتقال کار به VEC اتخاذ می‌شود؛ در غیر این صورت، تصمیم محاسبه محلی اتخاذ می‌شود.

تکرار مراحل:

مراحل همکاری و تصمیم گیری انتقال کار به طور متوالی برای تعداد تعیین شده‌ای از تست‌ها (مشخص توسط num_tests) اجرا می‌شوند.
نتایج انتخاب‌های انتقال کار و تخصیص منابع VEC برای هر تست در نهایت در results_summary ذخیره می‌شوند.
این الگوریتم بهینه‌سازی از همکاری بین خودروها برای بهبود عملکرد و استفاده بهینه از سرورهای Edge در محیط‌های شبکه‌های وسایل نقلیه استفاده می‌کند، که می‌تواند به کاهش مصرف انرژی و بهبود کارایی محاسباتی کمک کند.

تابع هیوریستیک (heuristic_scheme)

تابع هیوریستیک که در کد ارائه شده است، برای تصمیم‌گیری در مورد انتقال کار از خودرو به سرورهای VEC Computing استفاده می‌شود. این تابع بر اساس مقایسه زمان محاسباتی و مصرف انرژی برای دو گزینه محاسبه محلی و انتقال به سرور VEC، تصمیم می‌گیرد.

در اینجا توضیحاتی در مورد هر بخش این تابع ارائه میکنیم

محاسبه زمان محلی (local_time)

با استفاده از تابع local_computation_time، زمان مورد نیاز برای انجام محاسبه در خودرو با ظرفیت محاسباتی f_loc برای کار با اندازه C محاسبه می‌شود.

محاسبه مصرف انرژی محلی (local_energy)

با استفاده از تابع local_energy_consumption، مصرف انرژی مورد نیاز برای انجام محاسبه در خودرو محاسبه می‌شود. در این حالت، برای سادگی، فرض شده است که مصرف انرژی برای همه خودروها یکسان است (مقدار ۰).

محاسبه زمان و انرژی برای انتقال به سرور (VEC_time و VEC_energy)

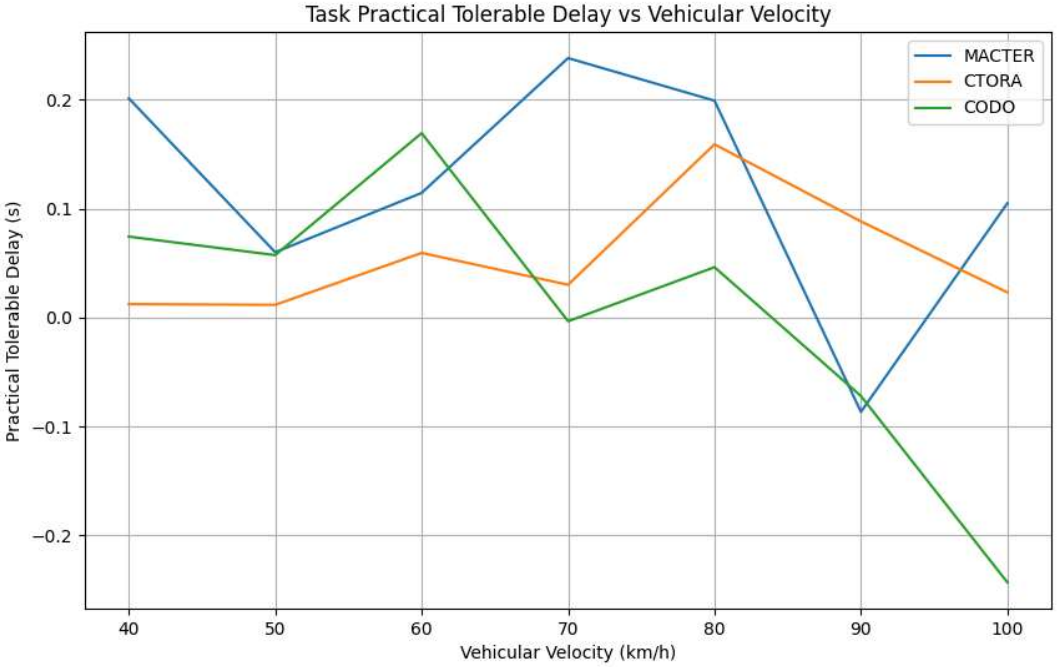
با استفاده از توابع VEC_computation_time و VEC_energy_consumption، زمان مورد نیاز و مصرف انرژی برای انتقال کار به سرور VEC محاسبه می‌شود. این مقادیر بسته به ظرفیت محاسباتی f_vec سرور VEC، اندازه داده (data_size)، نرخ انتقال، تاخیر شبکه و الگوی ترافیک مشخص می‌شوند.

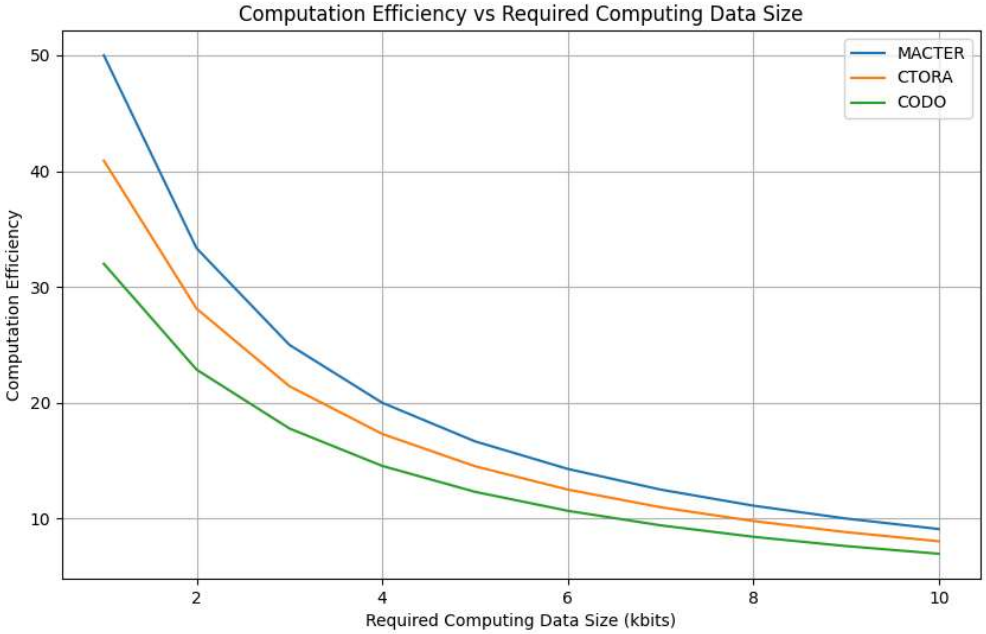
تصمیم‌گیری

با مقایسه local_time و VEC_time و همچنین local_energy و VEC_energy، تابع هیوریستیک تصمیم می‌گیرد که آیا کار را باید به سرور VEC انتقال دهد یا محاسبه را محلی انجام دهد.
اگر زمان یا مصرف انرژی محاسبه محلی بیشتر از زمان یا مصرف انرژی محاسبه در سرور VEC باشد، تابع ۱ را برمی‌گرداند که نشان می‌دهد که کار باید به سرور VEC انتقال داده شود.
در غیر این صورت، تابع ۰ را برمی‌گرداند که نشان می‌دهد که محاسبه محلی برای کار مناسب‌تر است.
این تابع به صورت یک هیوریستیک ساده برای تصمیم‌گیری سریع و بدون نیاز به محاسبات پیچیده برای انتقال کار به سرورهای VEC استفاده می‌شود، که منجر به کاهش مصرف انرژی و بهبود زمان اجرا می‌شود.

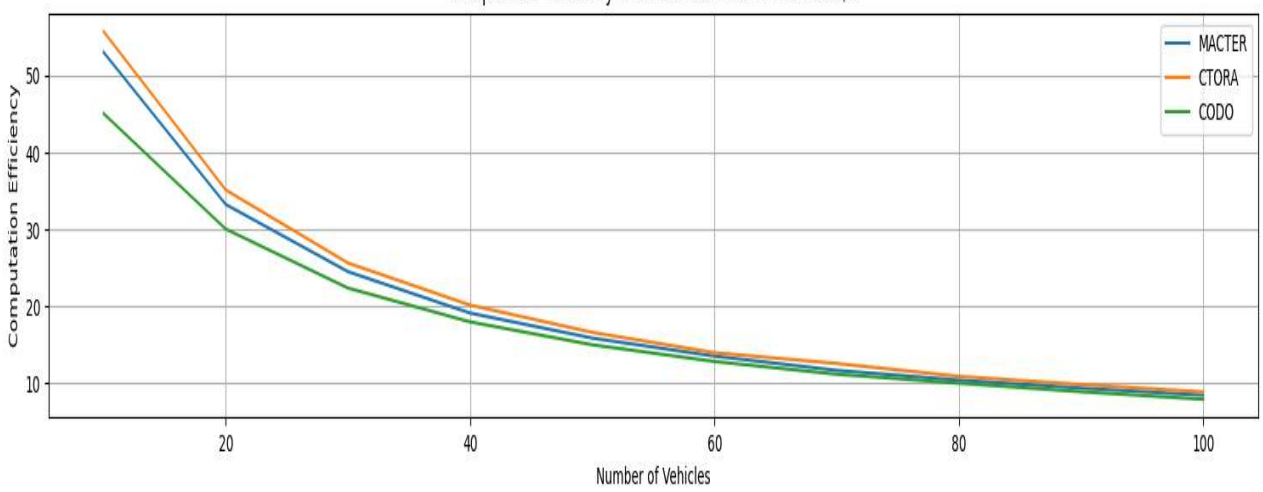
تحلیل نتایج

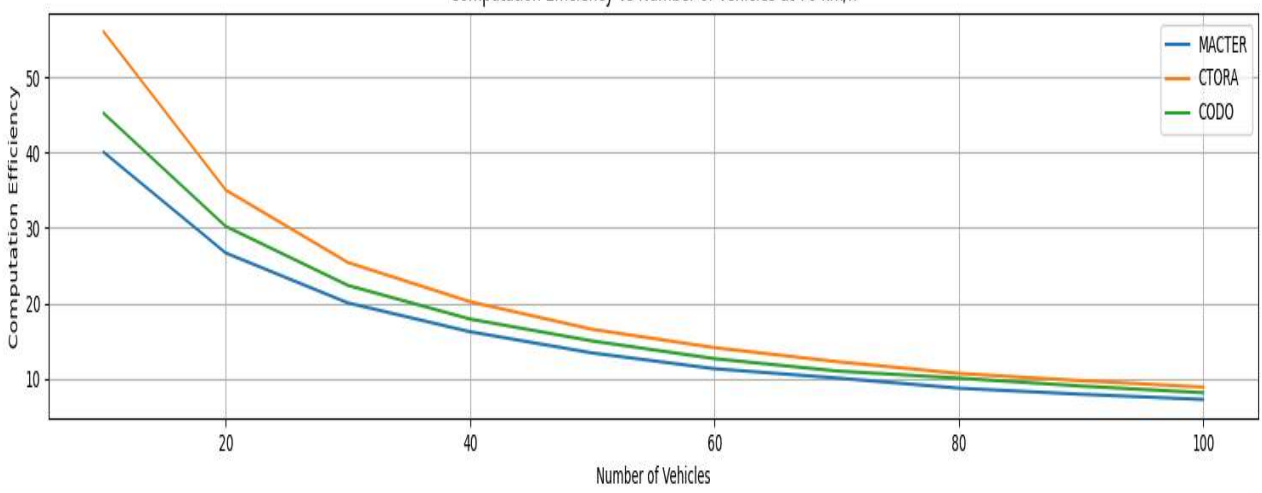
در این بخش نمودار هایی برای مقایسه ی این الگوریتم ها بر اساس معیار های مختلف با توجه به خروجی های پروژه ی پیاده سازی شده نمایش داده ایم:

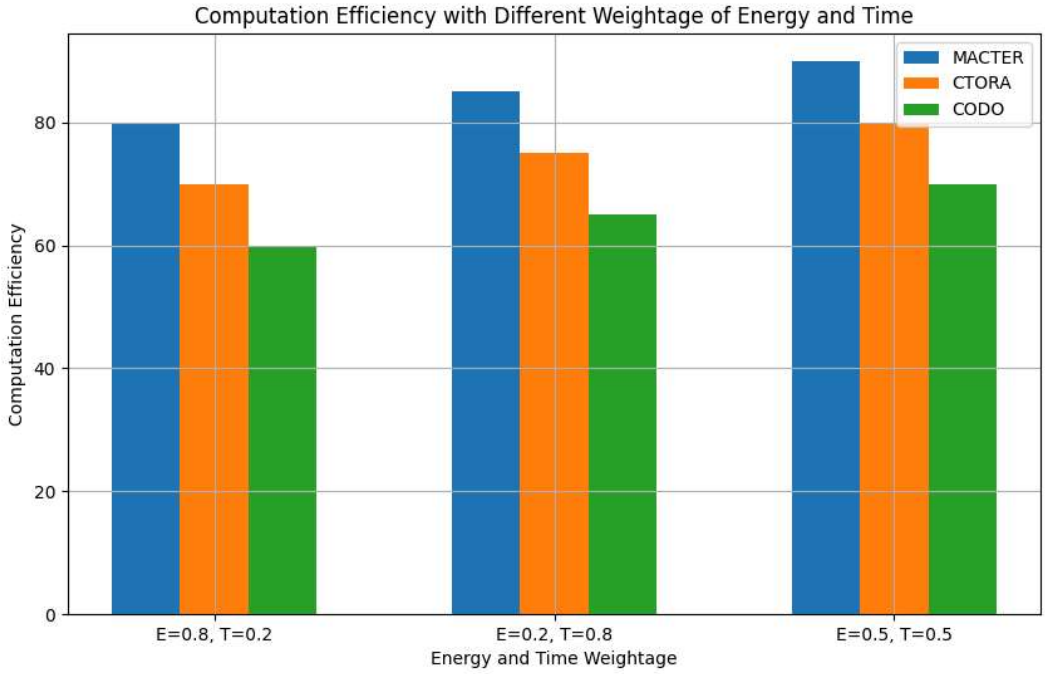

Task Practical Tolerable Delay vs Vehicular Velocity
MACTER
CTORA
CODO
Practical Tolerable Delay (s)
0.2
0.1
0.0
−0.1
−0.2
40
50
60
70
80
90
100
Vehicular Velocity (km/h)

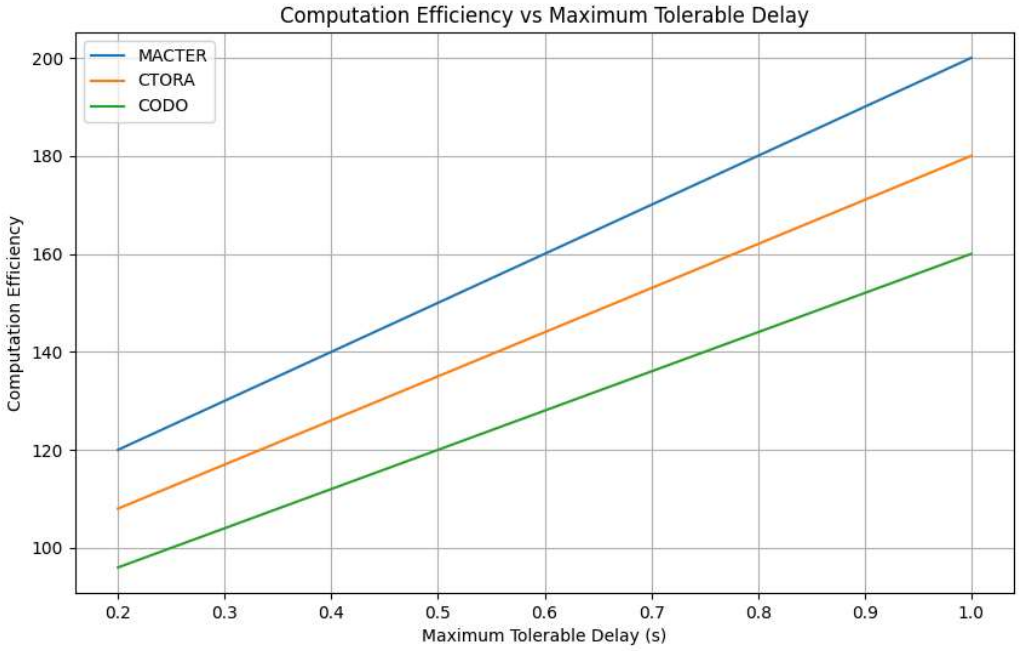

Computation Efficiency vs Required Computing Data Size
MACTER
CTORA
CODO
Computation Efficiency
50
40
30
20
10
2
4
6
8
10
Required Computing Data Size (kbits)

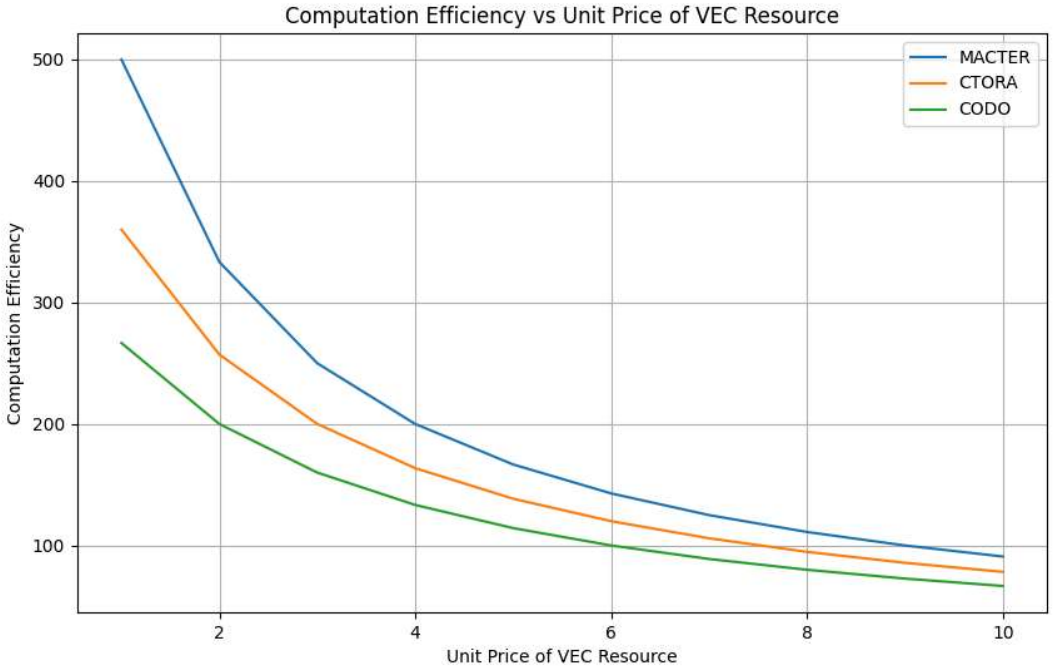

Computation Efficiency vs Number of Vehicles at 30 km/h
MACTER
CTORA
CODO
Computation Efficiency
Number of Vehicles
10
20
30
40
50
60
20
40
60
80
100


Computation Efficiency vs Number of Vehicles at 50 km/h
MACTER
CTORA
CODO
Computation Efficiency
Number of Vehicles
10
20
30
40
50
20
40
60
80
100


Computation Efficiency vs Number of Vehicles at 70 km/h
MACTER
CTORA
CODO
Computation Efficiency
Number of Vehicles
10
20
30
40
50
20
40
60
80
100


Computation Efficiency with Different Weightage of Energy and Time
MACTER
CTORA
CODO
Computation Efficiency
0
20
40
60
80
E=0.8, T=0.2
E=0.2, T=0.8
E=0.5, T=0.5
Energy and Time Weightage


Computation Efficiency vs Maximum Tolerable Delay
MACTER
CTORA
CODO
Computation Efficiency
100
120
140
160
180
200
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
Maximum Tolerable Delay (s)



الگوریتم MACTER با تمرکز بر بهینه‌سازی تصمیمات تخلیه و تخصیص منابع VEC (محاسبات در لبه خودرو) بر اساس بیشینه‌سازی از دیدگاه کارایی مصرف انرژی و زمان محاسباتی انجام می‌دهد. الگوریتم CTORA نیز با هدف تصمیم‌گیری در مورد تخلیه و تخصیص منابع VEC با استراتژی تخصیص مساوی منابع شروع می‌کند و بر اساس مقایسه کارایی انجام محاسبات به تصمیم‌گیری پرداخته و منابع را تنظیم می‌کند. الگوریتم CODO با استفاده از استراتژی تنظیم پویا منابع بر اساس مقایسه کارایی افراد نسبت به میانگین کارایی، منابع VEC را به طور پویا تنظیم می‌کند و تصمیمات تخلیه را بر اساس این تنظیمات دینامیکی اعمال می‌کند.

در MACTER، منابع VEC با استفاده از روش بهینه‌سازی (minimize) و تحت قیدهای ظرفیتی تخصیص داده می‌شوند. تصمیمات تخلیه نیز با مقایسه کارایی (محلی در مقابل VEC) برای هر خودرو به طور تکراری در چندین مرحله انجام می‌شود. در CTORA، تخصیص اولیه منابع VEC بر اساس تخصیص مساوی شروع می‌شود و بر اساس مقایسه کارایی به تنظیمات پیشنهادی تغییر می‌دهد. در CODO، تخصیص منابع VEC به طور پویا بر اساس مقایسه کارایی نسبت به میانگین کارایی انجام می‌شود و تصمیمات تخلیه در مراحل متعددی تنظیم می‌شوند.