远程协同打击攻防博弈挑战赛

智能体开发指南

中国运载火箭技术研究院

火箭军工程大学

南京大学

2025 年 8 月

目 录

一、概述

本文档为远程协同打击攻防博弈挑战赛智能体开发指南，内容包括场景想定、大模型部署运用、AI 开发框架及 AI 接口等信息，为各参赛队伍更好参与本次竞赛提供参考，本次比赛基于“决胜千里”智能博弈推演平台。

本次对抗以胡赛武装对海军编队实施非对称打击为背景。红方依托也门沿海丘陵地形隐蔽部署机动发射车，通过无人侦察飞行器定位目标，并发射高/低成本攻击弹实施饱和打击，同时由引导快艇前出提供末端制导以提升命中率。蓝方舰队沿亚丁湾国际航道航行，依托舰载雷达构建 50km 探测网，通过远程/近程拦截弹分层拦截，并调用舰载机反制红方快艇及发射车。战场动态性体现在：□海面能见度实时变化，直接影响红方侦察成功率；□随机生成的商船形成干扰源，威胁半径 500 米内攻击将触发扣分。

在上述背景下，本赛道通过大语言模型（LLM）凭借强大的语义理解与生成能力快速生成战前部署与想定（可用于推演），依托其涌现的博弈认知能力可模拟人类决策，再辅以大小模型协同，对海量策略进行高效探索，从而为军事智能决策开辟全新范式，推动军事智能技术在仿真推演领域的落地应用。

二、场景想定

（一）场景地图

（1）时空范围

本次比赛场景以胡赛武装对舰船编队集群实施非对称打击为背景，设定在吉布提外海曼德海峡、亚丁湾 600 公里海域范围（具体以实际地图标注为准）。红方依托沿海发射阵地实施机动打击，蓝方舰船编队集群沿亚丁湾国际航道航行。推演总时长设定为 1.5 小时，涵盖双方从战术机动、目标探测到火力打击的完整侦控打评流程。

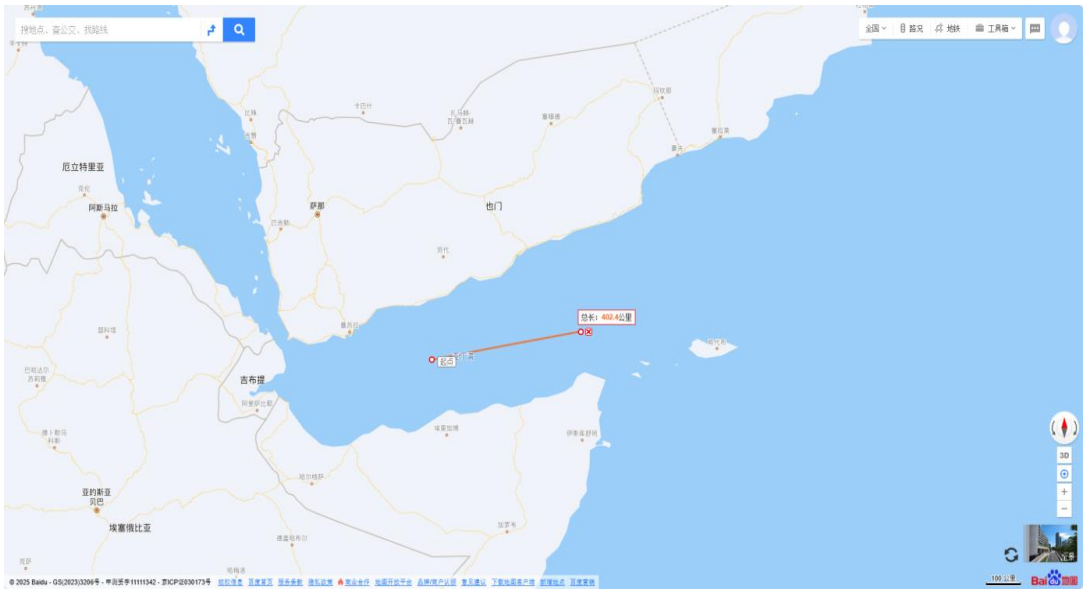

图 1 场景背景地图

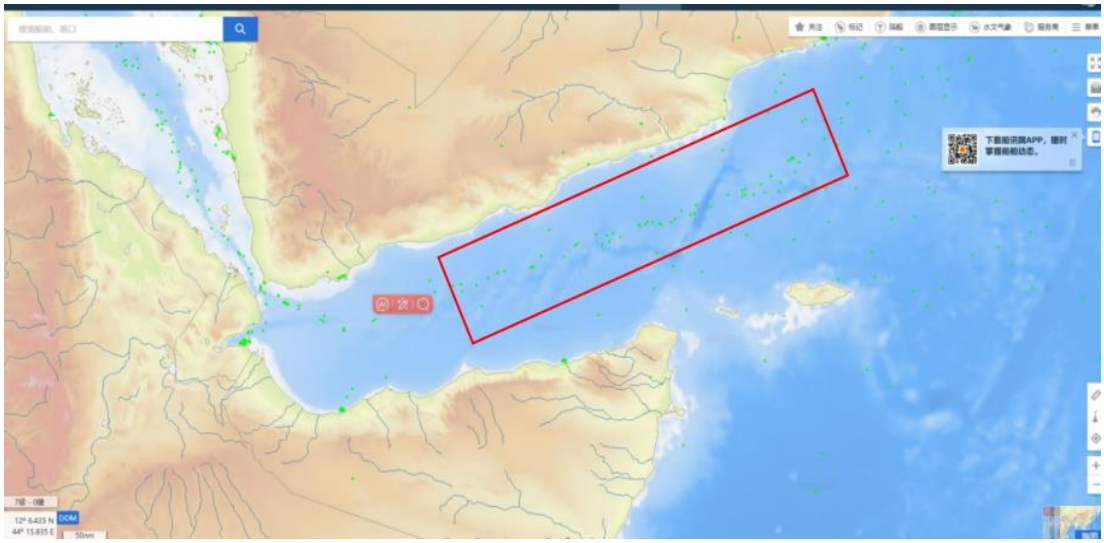

图 2 商船标注数据

（2）地图要素

平台地图采用瓦片式建模方式，考虑陆地和海洋信息。瓦片式地图即将地图按照分辨率分割成方形区域，对每块方形区域赋予相应的属性信息，以用于部署得分计算。

本次比赛场景中，地形主要考虑陆地地形和海洋区域，地形一方面可以通过 Python 开发接口获得，一方面可以通过 QGIS 软件进行查看，海洋上禁止部署机动发射车等模型，陆地上禁止部署旗舰、快艇、驱逐舰和巡洋舰。

（二）部署约束

本次对抗的核心区域位于吉布提外海（东经 43°-49°，北纬 12°-16°）。红方部署在也门沿海山地丘陵陆域（东经 45°-48°，北纬 14°-16°），要求发射车离散部署（≥5km 间距），红方可依托海岸丘陵隐蔽；蓝方部署在亚丁湾中部国际航线海域（东经 47°-49°，北纬 12°-14°），蓝方舰船初始部署时要远离商船（≥5km 间距）且舰船间有一定间距约束。

（一）红方部署约束

①机动发射车等装备初始部署在也门沿海（地图中也门境内，靠近红海一侧山地丘陵区），具体区间为东经 45°-东经 48°、北纬 14°-北纬 16° 陆域范围。

②发射车需分散布置，单车之间间隔不小于 5 公里（契合 “兵力分散、火力集中”思路），依托沿海地形（如丘陵、简易掩体 ）

隐蔽待机。

③引导快艇部署在也门沿海港口，初始在近岸海域（距也门海岸线 0 - 20 公里 ），即东经 43° - 东经 47° 、北纬 13° - 北纬 15° 海域待命，准备前出实施末端引导。引导快艇可目视观察自身周围 15km 内的敌方舰艇，并为高成本/低成本攻击飞行器提供末端引导，有效提高末制导命中概率。

（二）蓝方部署约束

①初始部署在亚丁湾中部偏东海域，即东经 47° -东经 49° 、北纬 12° -北纬 14° 范围；

②巡洋舰与旗舰间距不小于 30 公里，驱逐舰与旗舰间距不小于 20 公里，单艘驱逐舰之间间隔不小于 10 公里。

表 1 红蓝部署约束

名称	部署约束
红方	机动发射车等装备初始部署在也门沿海（地图中也门境内，靠近红海一侧山地丘陵区），具体区间为东经 45° -东经 48° 、北纬 14° -北纬 16° 陆域范围 。
	发射车需分散布置，单车之间间隔不小于 5 公里（契合 “兵力分散、火力集中” 思路），依托沿海地形（如丘陵、简易掩体 ）隐蔽待机。
	引导快艇部署在也门沿海港口，初始在近岸海域（距也门海岸线 0 - 20 公里 ），即东经 43° - 东经 47° 、北纬 13° - 北纬 15° 海域待命，准备前出实施末端引导。
蓝方	初始部署在亚丁湾中部偏东海域，即东经 47° -东经 49° 、北纬 12° -北纬 14° 范围；
	巡洋舰与旗舰间距保持 30 公里以上，驱逐舰与旗舰间距 20 公里以上，

	旗舰间最小间距在 20km 以上。
	蓝方初始部署时与商船的位置不小于 5 公里

（三）战场不确定设计

默认红蓝卫星实现了对核心区域全覆盖的侦察搜索，当舰船雷达静默时只能通过进攻方的侦察手段获取其具体位置，防御方同样无法侦察到飞行器实现多层拦截；当机动发射车隐蔽时则蓝方无法获取位置坐标，进攻方同样无法发射进攻武器对蓝方进行打击。

战场不确定设计：

①海面能见度：影响红方侦察设备的探测概率，探测概率与能见度成正比关系，如能见度为 1 时，探测概率为 100%；能见度为 0.5 时，探测概率 50%。考虑在实时推演时让海面能见度变成动态可变，仿真时间每小时变换一次。

②商船干扰：威胁半径 500 米触发扣分 50 分，其中，商船位置和数量每盘动态生成，在大模型部署时作为态势信息给到红蓝双方。

（四）红方要素

红方要素以“高低成本搭配+精准引导破防”为设计思路。

表 2 红方要素设计意图

名称	设计意图
机动发射车	实现打一枪换一地，通过隐蔽提高生存率。
高成本攻击弹（察打一体）	实现高价值目标斩首，探测半径 30km，对抗蓝方高价值目标，突防概率 60%，无引导命中率 70%，有引导命中率 90%。

低成本攻击弹	消耗拦截资源，饱和攻击迫使蓝方通道饱和，突防概率 30%，无引导命中率 40%，有引导命中率 60%。
引导快艇	对地对海探测 20km，对空探测 15km，提高末制导命中概率，提升高成本攻击弹/低成本攻击弹命中率 20%。
侦察无人机	低成本侦察飞行器探测半径 20km。

红方要素装备参数及能力如下表所示（非真实数据）：

表 3 红方要素装备参数及能力表

名称	抗毁伤能力	速度	射程	基础伤害值	探测目标类型	攻击目标类型	成本（工时）	备注
机动发射车	150	16m/s	/	/	/	/	4 周/车	
高成本攻击弹（察打一体）	40	1200m/s	600km	50	旗舰、巡洋舰、驱逐舰	旗舰、巡洋舰、驱逐舰	8 周/枚	30km 的探测半径，末制导半径 7km
低成本攻击弹	40	500m/s	300km	15	/	旗舰、巡洋舰、驱逐舰	3 周/枚	末制导半径 5km
引导快艇	80	18m/s	/	/	旗舰、巡洋舰、驱逐舰	/	2 周/艇	
侦察无人机	10	50m/s	/	/	旗舰、巡洋舰、驱逐舰、舰载机	/	0.5 周/架	

（五）蓝方要素

蓝方要素以“多层拦截+精准反击”为设计思路。

表 4 蓝方设计意图表

名称	设计意图
远程拦截弹	构成外层防御网，拦截高成本弹。
近程拦截弹	中层补防，拦截低成本弹群。
舰载机	动态反制关键节点。
电子战设备	开启后，形成最后一层防御策略，降低红方命中概率 20%。
舰对地巡航导弹	动态打击发射阵地的关键武器节点。

蓝方要素装备参数及能力如下表所示（非真实数据）。

表 5 蓝方要素装备参数及能力表（1）

名称	速度	抗毁伤能力	携带武器及数量	携带载荷	成本（工时）
旗舰	12m/s	200	挂载不超过 8 枚舰对地巡航导弹，冷却时间 180s	舰载雷达	24 周/船（损伤的维修成本与损伤的比例成正比，如损伤 50%需 12 周工时维修）
舰载机	250m/s	20	每架挂载 4 枚空空导弹，对无人侦察机进行打击；或挂载 2 枚空面导弹，对快艇和车进行打击	机载雷达	16 周/架（同上），考虑战备等级起飞总数量不超过 4 架。
驱逐舰	18m/s	100	远程/近程拦截弹*15，通道数为 2（2 个通道可同时拦截 2 个飞行器），冷却时间 180s；挂载不超过 16 枚舰对地巡航导弹，冷却时间 180s	舰载雷达	16 周/船（同上）
巡洋舰	15m/s	150	远程拦截弹*30，通道数为 2，冷却时间 180s；挂载不超过 12 枚舰对地巡航导弹，冷却时间	舰载雷达	12 周/船（同上）

			180s		

表 6 蓝方要素装备参数及能力表（2）

名称	抗毁伤能力	速度	射程	基础伤害值	探测目标类型	攻击目标类型	成本（工时）
近程拦截弹	/	1360m/s	20km	40	/	各类飞行器	1.5 周/枚
远程拦截弹	/	1360m/s	50km	40	/	各类飞行器	3 周/枚
空空导弹	/	1000m/s	10km	40	/	各类飞行器	4 周/枚
空面导弹	/	170m/s	10km	40	/	发射车、快艇	4 周/枚
舰对地巡航导弹	/	250m/s	200km	60	/	发射车、快艇	5 周/枚

表 7 蓝方要素探测设备参数及能力表

名称	挂载装置	数量	探测范围（半径）	探测目标类型
舰载雷达	舰载	/	50km(考虑了地形干扰超低空之后的修正值)	各类飞行器
对海瞭望	舰载	/	15km	商船、快艇
机载雷达	机载	/	对地对海 20km，对空 15km	无人机、商船、快艇

表 8 蓝方电子战设备参数及能力表

名称	挂载装置	数量	干扰范围（半径）	干扰目标类型
电子战设备	舰载	/	10km（每次开干扰可持续 5 分钟，冷却时间 10 分钟），干扰成功率 50%	无人机、弹、艇的探测/通信设备

（六）作战任务

（1）红方任务

① 攻击成本控制在 XX（具体数值由作战意图输入）以内；

② 至少 X 艘（具体数值由作战意图输入）蓝方驱逐舰/旗舰/巡洋舰失去作战能力（损伤≤30%）；

③ 在 X 分钟（具体数值由作战意图输入）内重创蓝方旗舰/巡洋舰/驱逐舰。

（2）蓝方任务

① 防御成本上限为 XX（具体数值由作战意图输入）；

② 保护旗舰不被击沉（旗舰损伤≥30%）；

③ 摧毁红方 X%（具体数值由作战意图输入）的车；

④ 拦截红方不少于 X%（具体数值由作战意图输入）攻击飞行器。

（3）作战意图输入参考案例

① 当海面能见度良好（信息不完全度 80%）时，红方以成本控制在 200 以内，根据战场情况调整攻击弹比例，争取在 3 小时内击沉蓝方旗舰，再对其他舰船进行有序打击。

② 设定防御成本上限为 200，蓝方依托舰载雷达和预警机，构建多层防御网，设定拦截成功率目标为 70% 以上，当发现红方飞行器来袭时，根据来袭目标数量和类型调整拦截弹发射数量，确保蓝方舰船受损率不超过 30%，保障舰队核心作战力量不受严重损失。

（七）推演规则

（1）机动规则

1）为了简化比赛，装备实体机动时仅考虑陆域、海域、空域的区分，即机动发射车仅能在陆地机动，考虑高程信息，快艇、驱逐舰、巡洋舰、旗舰仅能在海面机动，各类型弹、侦察无人机、舰载机仅能在海拔>0 的空域机动；

2）机动过程中的指令处理，装备实体机动指令下达后，仿真推演平台内会进行航迹解算，选手通过下达机动目标点位控制装备实体机动，同一帧针对同一装备实体选择最后一条指令，不同帧针对同一装备实体更新为最新一条指令；

3）装备实体机动范围限定在地图指定范围内；

4）机动状态转换：机动指令执行成功后，算子转换为机动状态；到达目标点后，转换为停止状态。

（2）隐蔽规则

1）机动发射车由机动状态转为隐蔽状态，状态转换时间为 180s；

2）若机动发射车处于隐蔽、机动状态，则需接收 AI 指令进行状态转换；

3）若机动发射车处于隐蔽状态或状态转换过程中，无法被蓝方探测，但也不能发射武器装备。

（3）发射规则

1）装备实体发射不同武器设备需要考虑冷却时间，通过 AI 接口指令控制，若在冷却时，平台接收到发射指令，将不予执行；

2）机动发射车发射高成本打击弹的冷却时间为 10 分钟，认为是装载导弹的时间；

3）机动发射车发射低成本打击弹的冷却时间为 10 分钟，认为是装载导弹的时间；

4)旗舰/驱逐舰/巡洋舰发射舰对地巡航导弹的冷却时间为 180s；

5）驱逐舰发射通道数为 2，可同时发射 2 枚远程/近程拦截弹，每个通道的发射冷却时间为 180s；

6）巡洋舰发射通道数为 2，可同时发射 2 枚远程拦截弹，每个通道的发射冷却时间 180s。

（4）携带约束规则

1）机动发射车、旗舰、舰载机、驱逐舰、巡洋舰有最大携带武器数量约束，超出数量范围的武器挂载将会失效；

2）旗舰每艘挂载不超过 8 枚舰对地巡航导弹；

3）舰载机每艘挂载不超过 4 枚空空导弹，或挂载不超 2 枚空面导弹，空空导弹和空面导弹只能二选一；

4）驱逐舰每艘挂载远程/近程拦截弹不超过 15 枚，同时挂载不超过 16 枚舰对地巡航导弹；

5）巡洋舰每艘挂载不超过 30 枚远程拦截弹，同时挂载不超过 12 枚舰对地巡航导弹；

6）机动发射车上挂载的高成本攻击弹或者低成本攻击弹受到总成本约束，发射车通过简化认为可以更新弹药，不设置挂载数量限制。

（5）探测裁决规则

1）探测有效原则：一是目标在观察范围内，二是满足探测识别概率要求；

2）红方高成本攻击弹（察打一体）探测半径 30km，侦察无人机探测半径 20km，引导快艇的探测半径 15km；

3）蓝方舰载雷达探测半径 50km（针对各类飞行器），对海瞭望 15km（针对快艇、商船），舰载机对地对海 20km，对空 15km；

4）探测识别概率受海面能见度影响，与能见度成正比关系，如能见度为 1 时，探测概率为 100%；能见度为 0.5 时，探测概率 50%；

5）本次比赛场景不设置红蓝卫星，默认对核心区域进行全覆盖侦察搜索，当舰船雷达静默时只能通过进攻方的侦察手段获取其具体位置，防御方同样无法侦察到飞行器实现多层拦截；当机动发射车隐蔽时则蓝方无法获取位置坐标。

6）旗舰、驱逐舰、巡洋舰和快艇探测设备开关机由 AI 控制，无人机、高成本进攻弹和舰载机的探测设备默认全程开机，不设置探测时长约束；

（6）干扰裁决规则

1）干扰有效原则：一是目标在干扰范围内，二是满足干扰成功率判定要求；

2）电子战设备仅能部署到驱逐舰和旗舰上；

3）电子战设备干扰范围为 10km；

4）电子战设备每次开干扰可持续 5 分钟，冷却时间 10 分钟，干扰成功率 50%；

5）电子战设备主要针对红方无人机、弹、艇的探测/通信设备。

（7）毁伤裁决规则

1）毁伤有效原则：一是目标在毁伤范围内，二是满足命中概率要求；

2)依据飞行航迹和目标运动航迹，当两者绝对位置小于 50 米（对舰）、10 米（对地、小艇）5 米（对空）时，根据命中概率判定是否击中；

3）毁伤程度判定：当判定为击中时，命中目标扣除基础毁伤数值，直到为 0 为止；

4）毁伤累积原则：目标毁伤程度采用线性累加方式统计，以毁伤累积大于 100%为最终目标；

5）高成本攻击弹打击旗舰/驱逐舰/巡洋舰无引导命中概率 70%，

有引导命中率 90%；

6）低成本攻击弹打击旗舰/驱逐舰/巡洋舰无引导命中率 40%，有引导命中率 60%；

7）引导快艇提高末制导命中概率，提升高成本攻击弹/低成本攻击弹命中率 20%；

8）空空导弹命中概率为 40%；

9）空面导弹命中概率对引导快艇 60%，对机动发射车命中概率 45%；

10）舰对地巡航导弹命中概率为 60%。

（8）拦截裁决规则

1）拦截有效原则：一是目标在拦截范围内，二是满足拦截概率要求；

2）拦截规则：拦截弹飞行至与目标交汇点，当距离小于 50m；

3）低成本攻击弹被远程/近程拦截弹的拦截概率 70%；

4）高成本攻击弹被远程/近程拦截弹的拦截概率 40%；

5）侦察无人机被远程/近程拦截弹的拦截概率 85%。

（八）得分规则

通过以最小代价最大化对手损失并完成战略目标核心诉求，兼顾战术毁伤、任务达成与经济消耗，构建公式如下：

总分 = 毁伤得分 + 任务得分 - 经济惩罚 - 违规扣分

（1）毁伤得分

在仿真推演过程中，红方对蓝方旗舰、驱逐舰、巡洋舰目标进行火力打击，击中后根据毁伤裁决规则后的损伤比例计分，其中，旗舰150 分/艘、驱逐舰 50 分/艘、巡洋舰 80 分/艘，如下表所示。

表 9 红方对蓝方目标毁伤得分表

目标类型	分值	说明
旗舰	150 分/艘/ 按损伤比例计分	击沉
驱逐舰	50 分/艘/ 按损伤比例计分	击沉
巡洋舰	80 分/艘/ 按损伤比例计分	击沉

蓝方对红方机动发射车、引导快艇进行打击，通过毁伤裁决规则判定击中后按损伤比例计分，对高成本攻击弹、低成本攻击弹、侦察无人机等进行拦截，通过拦截裁决规则后进行计分。

表 10 蓝方对红方目标毁伤得分表

目标类型	分值	说明
机动发射车	60 分/辆/ 按损伤比例计分	击毁
引导快艇	20 分/艘/ 按损伤比例计分	击沉
高成本攻击飞行器	8 分/枚	拦截
低成本攻击飞行器	4 分/枚	拦截
侦察无人机	1 分/架	拦截

备注：

①击伤未沉没：按目标分值×损伤比例（如驱逐舰损伤 70%得

35 分）

（2）任务得分

红蓝均有任务得分，任务得分是以推演前由裁判员输入的作战意图为依据，根据推演结束后的推演结果数据分析，判断红蓝双方任务完成情况，若全部完成，得 200 分，若部分完成（50%~99%达成目标），得 100 分，若失败，则得 0 分。

表 11 任务得分表

任务状态	分值	条件
完成	200	100%达成目标
部分完成	100	50%~99%达成目标
失败	0	<50%

（3）经济惩罚

红方总经济消耗C_{red}：所有发射的飞行器使用成本（含被拦截/未命中飞行器）+机动发射车/快艇战损成本（若被摧毁）；

蓝方总经济消耗C_{blue}：拦截弹使用成本+舰载机使用成本+舰船战损维修成本；

红方经济惩罚：当$\frac{C_{red}}{C_{blue}}$ >1.5，每超过 10%，扣 10 分，上限为 100 分，即消耗超过蓝方 50%以上才扣分，低于则不惩罚，若目标完成，惩罚减半；

蓝方经济惩罚：当$\frac{C_{blue}}{C_{red}}$ >1.2时，每超过 10%，扣 10 分，上限为 100 分，即消耗超过红方 20%以上才扣分，低于则不惩罚。

（4）违规扣分

①商船保护：双方武器威胁到商船时，攻击者扣 50 分/艘，武器命中商船周围半径 500 米处，即视为威胁到商船；

②发射车分散：部署发射车间距小于 5 公里，扣 10 分/个；

③舰船部署分散：当巡洋舰与旗舰间距小于 30 公里，驱逐舰与旗舰间距小于 20 公里，单艘驱逐舰之间间隔小于 10 公里，扣 10 分/个。

三、AI 开发及大模型运用指南

（一）智能博弈决策点

本次专项赛道，选手需要开发一套基于大模型提示词驱动的智能体。智能体需能够自主调用推演平台接口工具，完成战前场景部署以及战中临机决策两大核心阶段。选手开发重点从编写复杂兵棋推演逻辑代码，转向设计大模型提示词与智能体自动化执行流程，引导大模型智能体能够自主完成战前场景部署与战中任务执行。

表 12 战前场景部署决策点表

序号	红蓝部署决策点
1	挂载装备选择
2	挂载装备数量生成
3	装备数量生成
4	装备的位置部署

表 13 红方战中临机决策决策点表

序号	红方决策点
1	发射车状态转换

2	发射车在线机动
3	侦察无人机在线机动
4	引导快艇在线机动
5	低成本攻击弹目标分配
6	高成本攻击弹目标分配

表 14 蓝方战中临机决策决策点表

序号	蓝方决策点
1	旗舰/巡洋舰/驱逐舰在线机动
2	旗舰/巡洋舰/驱逐舰雷达开关机
3	巡洋舰/驱逐舰拦截分配
4	旗舰/巡洋舰/驱逐舰火力分配
5	舰载机在线机动
6	舰载机火力分配
7	旗舰/巡洋舰/驱逐舰干扰开机

（二）AI 开发总体介绍

“决胜千里”智能博弈推演平台提供大模型战前场景部署开发所需的场景编辑工具和大模型战中临机决策开发所需的仿真引擎，在大模型战前场景部署开发环节中，选手依据大模型智能开发框架，专注于开展大模型选型、提示词编写、知识库构建和新增接口 MCP 封装等工作，红蓝双方执行完成后，依据“决胜千里”内置的大模型工具将其转换为可用于仿真推演的场景想定；在大模型战中临机

决策开发环节中，选手依据大模型智能开发框架，专注于红蓝小模型开发、提示词编写、大小模型协同等工作，在训练状态下，若小模型采用强化学习等算法，可提供红蓝部分奖励信息，开发框架将态势等信息转发给智能体，智能体形成 AI 决策指令转发给仿真平台开始执行，如下图所示。

图 3 AI 开发总体框架

（三）AI 开发框架文件系统说明

决胜千里大模型智能体开发框架封装了决胜千里编辑器与仿真引擎的连接、封装了实体部署、态势感知、决策洗发等指令，提供了大模型调用工具，提示词等定义框架并内置了相关工具和提示词，

整体框架如图 1 所示。为选手提供了基于决胜千里平台的 Python 开发大模型智能体开发环境，选手可以专注于基于的战前兵力部署和战中临机决策算法设计。在赛前阶段，选手基于本开发框架和指南开发好智能体后，将智能体迁移到比赛环境中。

本项目提供的压缩包，包含 jsqlsim pythonSDK 包，JSQLSceneTool， JSQLEngine 三个文件。

（1）jsqlsim-0.1.0-py3-none-any.whl 为本项目 python sdk，用户通过改 SDK 和决胜千里编辑器（JSQLSceneTool)，决胜千里推演引擎(JSQLEngine) 交互。

（2）决胜千里编辑器（JSQLSceneTool)，场景编辑器，用户通过 python sdk 和该编辑器交互，进行兵力部署。

（3）决胜千里推演引擎(JSQLEngine), 用户通过 python sdk 进行战中决策。

（四）大模型部署和运用

SDK 提供了 BaseOpenAIAgent 类，用户可以以此为基类，在初始化构造时候，传入 API 的地址，api_key，使用的模型，使用方式如下所示。

```
from jsqlsim.agents import BaseOpenAIAgent
from jsqlsim.llm import tools_repo

class Demo(BaseAgent):
    def __init__(self, base_url, api_key, model, tools=None, tools_registry=None):
        super().__init__(base_url, api_key, model, tools, tools_registry)
```

```
    def run(self, prompt):
        return self.chat(prompt, system_prompt=None)


if __name__ == "__main__":
    tools = tools_repo.get_tools()
    tools_registry = tools_repo.get_registry()
    agent = Demo(
        base_url="https://api.siliconflow.cn/v1",
        api_key="XXX",
        model="Qwen/Qwen3-14B")
    result = agent.run("计算半径为 10 的圆的面积")
    print(result)
    result = agent.run("无人侦察弹的装备信息")
    print(result)
    agent.clear_history()
    result = agent.run("计算 1 个高成本打击弹和 2 个侦察无人机的打击成本")
    print(result)
```

用户可以通过下述示例方式，使用 prompts_repo 注册工厂，通过 get_prompts 方法获取所有可使用提示词列表，通过 get 方法获取指定名称的可调用提示词对象，通过 get_registry 方法获取所有提示词的注册表。

```
from typing import Dict, List

from jsqlsim.llm import Prompt, prompts_repo

# 1. 获取所有提示词
prompts: List[Prompt] = prompts_repo.get_prompts()
```

```
# 2. 获取装备标签下的所有提示词
equipment_prompts: List[Prompt] = prompts_repo.get_prompts(tags=["装备编配"])

# 3. 获取提示词实例
force_compose_prompt_template: Prompt = prompts_repo.get("force_compose_prompt_template")
result = force_compose_prompt_template(budget=20, target_ships=4, equipment_summary="", deployment_constraints="")
print(result)

# 4. 获取所有提示词
prompts: Dict[str, Prompt] = prompts_repo.get_registry()

# 5. 获取提示词实例
spec_tool: Prompt = prompts.get("force_compose_prompt_template")
# 调用提示词示例对象
result = force_compose_prompt_template(budget=20, target_ships=4, equipment_summary="", deployment_constraints="")
print(result)
```

大模型调用工具示例如下，通过 sdk 内置的 BaseOpenAIAgent 调用预制或者自定义的工具。

```
from jsqlsim.agents import BaseOpenAIAgent
from jsqlsim.llm import tools_repo

@tools_repo.register(
    parameters={
        "type": "object",
        "properties": {
            "x": {"type": "number", "description": "第一个数字"},
            "y": {"type": "number", "description": "第二个数字"}
        },
```

```
            "required": ["x", "y"]
        }
)
def add_numbers(x: float, y: float) -> float:
    """计算两个数字的和"""
    return x + y

class Demo(BaseAgent):
    def __init__(self, base_url, api_key, model, tools=None, tools_registr
y=None):
        super().__init__(base_url, api_key, model, tools, tools_registry)

    def run(self, prompt):
        return self.chat(prompt, system_prompt=None)



if __name__ == "__main__":
    tools = tools_repo.get_tools()
    tools_registry = tools_repo.get_registry()
    agent = Demo(
        base_url="https://api.siliconflow.cn/v1",
        api_key="sk-aybnsyqfbcpdprxewwilroegaijrotjqxnxsdvbvvttciwer",
        model="Qwen/Qwen3-14B")
    result = agent.run("计算半径为 10 的圆的面积")
    print(result)
    result = agent.run("无人侦察弹的装备信息")
    print(result)
    agent.clear_history()
    result = agent.run("计算 1 个高成本打击弹和 2 个侦察无人机的打击
成本")
    print(result)
```

用户可以通过下述示例方式，使用 tools_repo 注册工厂，通过 get_tools 方法获取所有可使用工具列表，通过 get 方法获取指定名称

的可调用工具对象，通过 get_registry 方法获取所有工具的注册表。

```
from typing import Dict, List

from jsqlsim.llm import Tool, tools_repo

# 1. 获取所有工具
tools: List[Tool] = tools_repo.get_tools()

# 2. 获取装备标签下的所有工具
equipment_tools: List[Tool] = tools_repo.get_tools(tags=["装备"])

# 3. 获取工具实例
spec_tool: Tool = tools_repo.get("get_equipment_specs")
# 查询存在的装备
equipment_name = "高成本打击弹"
result = spec_tool(equipment_name=equipment_name)
print(result)

# 4. 获取所有工具
tools: Dict[str, Tool] = tools_repo.get_registry()

# 5. 获取工具实例
spec_tool: Tool = tools.get("get_equipment_specs")
# 查询存在的装备
equipment_name = "高成本打击弹"
result = spec_tool(equipment_name=equipment_name)
print(result)
```

sdk 内置了一系列大模型可以使用的工具/提示词，包含战前部署和战中临机决策的，具体工具/提示词名称和使用说明见下文。

（五）大模型战前场景部署开发与调试

（1）配置开发环境

为确保开发环境的隔离性和可移植性，推荐使用 uv 工具进行 Python 依赖管理。uv 是一个现代化的 Python 包管理器，兼具速度快、可复现等优势。

Windows 下安装 uv，并启动 uv 环境：

步骤 1：安装 uv

打开管理员权限的 PowerShell 终端，执行以下命令：

```
powershell -ExecutionPolicy ByPass -c "irm https://astral.sh/uv/install.ps1 | iex"
```

安装完成后，重启终端，输入以下命令验证是否成功：

```
uv --version
```

步骤 2：创建并启动 uv 虚拟环境

进入项目目录

```
cd your_project_path
```

创建虚拟环境：

```
uv venv
```

激活虚拟环境：

```
.venv\Scripts\Activate.ps1
```

Linux 下安装 uv，并启动 uv 环境：

步骤 1：安装 uv

在终端中执行以下命令：

```
curl -LsSf https://astral.sh/uv/install.sh | sh
```

安装完成后，重启终端，验证安装：

```
uv --version
```

步骤 2：创建并启动 uv 虚拟环境

进入项目目录

```
cd your_project_path
```

创建虚拟环境：

```
uv venv
```

激活虚拟环境：

```
source .venv/bin/activate
```

（2）启动

启动平台场景编辑工具后进入下面界面，启动后 GRPC 服务自启动：

图 4 平台场景编辑工具

（3）创建智能体开发工程（打开 AI Demo）

我们提供了简单的 AI Demo，供选手改造，下面是 AI Demo 部分开发代码：

```
import json
from jsqlsim.agents.base import BaseOpenAIAgent
from jsqlsim.llm.tools import equip_tool_registry


class EquipMountAgent(BaseOpenAIAgent):
    def __init__(self, base_url, api_key, model):
        super().__init__(base_url, api_key, model)
        ## SS: 这里先用硬编码了，后续可以改造成 MCP
        self.tool_registry = equip_tool_registry
```

```
    def mount_equipment(self, plan, mount_constraints):
        # 自动构建 prompt（如果注册了）
        if self.prompt_func:
            user_prompt = self.prompt_func(
                selected_plan=plan,
                mount_constraints_dict=mount_constraints,
            )
        else:
            raise ValueError("未设置 prompt_func，请先调用 agent.use(prompts=...)")
        print("user_prompt: ", user_prompt)
        response = self.chat(user_input=user_prompt)
        print("response: ", response)
        return json.loads(response)
    def auto_mount_validation(
        self,
        mount_result,
        selected_plan,
        mount_constraints,
        max_retry=3,
        verifier_func=None
    ):
        if verifier_func is None:
            raise ValueError("请传入验证函数 verify_mount_plan")
```

```
        retry = 0
        extra_feedback = None
        while retry < max_retry:
            # 校验挂载方案
            validation_result = verifier_func(
                mount_result=mount_result,
                selected_plan=selected_plan,
                mount_constraints=mount_constraints
            )
            if validation_result["success"]:
                print(f"挂载方案验证通过（第 {retry + 1} 次尝试）")
                return mount_result
            else:
                # 有反馈信息则继续 retry，否则终止
                extra_feedback = validation_result.get("feedback", None)
                if not extra_feedback:
                    print("验证失败，但未提供可用反馈信息，终止重试。
")
                    return mount_result
                mount_result = self.chat(user_input=extra_feedback)
                mount_result = json.loads(mount_result)

                print(f" 第 {retry + 1} 次 尝 试 失 败 ， 反 馈 信 息 ：
{extra_feedback}")
                retry += 1
```

```
        print("超过最大重试次数，返回最后一次挂载结果")
        return mount_json
```

下面是一个大模型场景部署智能体的部分提示词，选手可以在此基础上修改：

```
mount_equipment_prompt_template = Template("""
你是一名红方作战装备编配专家，任务是将已选定的装备数量，分配成具体的实体单元（挂载关系），不要过度思考，尽可能快的直接输出结果。

你收到的任务，是根据**第一阶段装备规划结果**，按照挂载约束分配具体挂载关系。但是如果你发现第一阶段分配不合理（例如平台数量不足、弹药无法挂载等），你需要：

1. **先尝试合理挂载**当前装备；
2. 如果明显不合理（如平台不够、挂载超载），**请返回 revision_required = true，并给出文字版调整建议**；
3. 不要强行输出错误挂载方案，优先保证实际可执行性；
4. 调整建议不要直接输出修改后的装备清单，而是输出一句完整的话，例如：“当前弹药数量无法挂载，请至少增加 1 辆机动发射车”。

---

第一阶段规划输出如下（你可以认为它是建议，可以被修正）：

${selected_plan}

---

红方平台挂载约束如下：

${mount_constraints}

---
```

请输出结构化 JSON，输出格式如下：

```
{
  "revision_required": false,
  "entities": [
    {
      "type": "平台名称",
      "mounts": {
        "装备名称 A": 数量,
        "装备名称 B": 数量
      }
    },
    {
      "type": "独立平台名称",
      "mounts": {}
    }
  ],
  "reason": "挂载安排说明"
}
# 如果 revision_required = true（挂载不合理），则改为如下格式输出：
{
  "revision_required": true,
  "revision_suggestion": "当前弹药数量无法挂载，请至少增加 1 辆机动发射车。",
  "reason": "当前挂载方案不可行，存在平台不足。"
```

```
}
""")


def format_plan_to_prompt_text(plan: dict) -> str:
    platform_lines = []
    equipment_lines = []
    selected_platform = plan.get("selected_platform", {})
    selected_equipment = plan.get("selected_equipment", {})
    if selected_platform:
        for idx, (name, count) in enumerate(selected_platform.items(), 1):
            platform_lines.append(f"  {idx}、{name}，数量: {count}")
    if selected_equipment:
        for idx, (name, count) in enumerate(selected_equipment.items(), 1):
            equipment_lines.append(f"  {idx}、{name}，数量: {count}")

    platform_block = "\n".join(platform_lines)
    equipment_block = "\n".join(equipment_lines)
    return f"""选定的平台：
{platform_block}
选定的装备：
{equipment_block}"""


def format_mount_constraints(mount_constraints: dict) -> str:
    lines = []
```

```
    for idx, (platform, info) in enumerate(mount_constraints.items(), start=1):
        allowed_equipment = info.get("可挂载装备", [])
        extra_constraints = info.get("挂载约束", [])
        # 判断“无需挂载”类平台
        if allowed_equipment == ["无需挂载装备"] or not allowed_equipment:
            lines.append(f"    {idx}、{platform}，无需挂载装备")
        else:
            equip_list = "、".join(allowed_equipment)
            line = f"    {idx}、{platform}，支持挂载装备：{equip_list}"
            # 格式化附加约束为（1）（2）编号形式
            if extra_constraints:
                constraints_list = "；".join(
                    f"（{i+1}）{c}" for i, c in enumerate(extra_constraints)
                )
                line += f"\n        - 附加约束：{constraints_list}"
            lines.append(line)
    return "\n".join(lines)


def render_mount_equipment_prompt(
    *, selected_plan: dict, mount_constraints_dict: dict)-> str:
    selected_plan_str = format_plan_to_prompt_text(selected_plan)
    mount_constraints_text = format_mount_constraints(mount_constraints_dict)
```

```
    return mount_equipment_prompt_template.substitute(
        selected_plan=selected_plan_str,
        mount_constraints=mount_constraints_text,
    )
```

（4）调试与运行智能体

为保障逻辑正确性和大模型输出质量，推荐使用 pytest 在每个阶段编写单元测试，例如：

```
pytest -s tests/test_equip_mount.py
```

单元测试脚本，可以构造每个阶段的临时数据，进行测试，例如：

```
# --------------------------
#  测试数据准备
# --------------------------
MOUNT_CONSTRAINTS = {
    "机动发射车": {
        "可挂载装备": ["高成本打击弹", "低成本打击弹", "无人侦察弹", "无人干扰弹"],
        "挂载约束": ["发射车没有挂载装备的数量限制，但是每辆发射车必须挂载装备", "每辆发射车发射一次后，需要 180 秒冷却才能再次发射"]
    },
    "引导快艇": {
        "可挂载装备": ["无需挂载装备"],
        "挂载约束": []
```

```
    },
    "侦察无人机": {
        "可挂载装备": ["无需挂载装备"],
        "挂载约束": []
    }
}

SELECTED_PLAN = {
    "selected_platform": {
        "机动发射车": 2,
        "侦察无人机": 1
    },
    "selected_equipment": {
        "高成本打击弹": 4,
        "低成本打击弹": 1,
        "无人侦察弹": 1,
        "无人干扰弹": 1,
    },
    "reason": "考虑到预算与挂载能力，选择一辆发射车携带 4 枚打击弹并配备一架侦察无人机。"
}

def verify_mount_func(mount_result: dict, selected_plan: dict, mount_constraints: dict) -> dict:
    """
```

```
    验证挂载方案是否合法，返回是否通过验证及额外反馈。
    """
    # 快速跳过 revision_required=true 的情况（LLM 自己就判断不合理）
    if mount_result.get("revision_required", False):
        return {
            "success": False,
            "feedback": mount_result.get("revision_suggestion", "模型判断当前挂载方案不合理，请重新规划。")
        }
    selected_platform = selected_plan.get("selected_platform", {})
    selected_equipment = selected_plan.get("selected_equipment", {})
    entities = mount_result.get("entities", [])
    # 统计实体中平台和装备挂载总数
    platform_counter = {}
    equipment_counter = {}
    for entity in entities:
        platform_type = entity["type"]
        platform_counter[platform_type] = platform_counter.get(platform_type, 0) + 1

        for eq, count in entity.get("mounts", {}).items():
            equipment_counter[eq] = equipment_counter.get(eq, 0) + count

            # 校验挂载是否允许
            allowed_equips = mount_constraints.get(platform_type, {}).get("可挂
```

```
载装备", [])
                if eq not in allowed_equips:
                    return {
                        "success": False,
                        "feedback": f"平台「{platform_type}」不允许挂载「{eq}」，请重新调整挂载关系。"
                    }

    # 平台数量验证
    for plat, expected_count in selected_platform.items():
        actual_count = platform_counter.get(plat, 0)
        if actual_count > expected_count:
            return {
                "success": False,
                "feedback": f"平台「{plat}」使用数量超过规划上限（最多{expected_count}台），请减少该平台数量。"
            }

    # 装备数量验证
    for eq, expected_count in selected_equipment.items():
        actual_count = equipment_counter.get(eq, 0)
        if actual_count != expected_count:
            return {
                "success": False,
                "feedback": f"装备「{eq}」挂载数量为 {actual_count}，但规
```

```
划中数量为 {expected_count}，请修正挂载分配。"
                }
    # 一切正常
    return {
        "success": True
    }


def test_mount_compose_equipment():
    agent = EquipMountAgent(
        base_url="xxxxxx",  ## 填入在线大模型 base_url
        api_key=" xxxxxx ",  ## 填入在线大模型 api_key
        model="xxxxxx")  ## 填入在线大模型名称
    agent.use(prompts=render_mount_equipment_prompt, tools=[])
    mount_result = agent.mount_equipment(
        plan=SELECTED_PLAN,
        mount_constraints=MOUNT_CONSTRAINTS,
    )
    print("mount_result: ", mount_result)
    ## 如果第一阶段规划合理，则执行挂载关系正确性验证
    if not mount_result.get("revision_required"):
        mount_result = agent.auto_mount_validation(
            mount_result=mount_result,
            selected_plan=SELECTED_PLAN,
            mount_constraints=MOUNT_CONSTRAINTS,
```

```
verifier_func=verify_mount_func
```

（六）大模型战中临机决策开发与调试

（1）配置开发环境

步骤 1：以 Windows 环境为例，推荐安装 Anaconda 以创建虚拟环境，安装 VScode 编辑器及相应 Python 插件以方便调试。安装 git 版本管理工具以便进行版本管理。

步骤 2：创建虚拟环境并安装所需的常用模块如 numpy、pytorch 等。

```
conda create --name <your_env_name>

conda install <your_model>
```

推荐的运行环境为 python 3.12.4，并有以下包，其他可参考 requirements.txt，暂未发现由于包版本不同导致的兼容性问题。GPU 版本 torch 要求 CUDA 环境，从简化配置角度起见，可选择 CPU 版本。

numpy==1.26.0	pandas==2.2.2
scikit-learn==1.5.1	torch==2.4.0+cu118
scipy==1.14.1	torchvision==2.4.0+cu118
inspyred==1.0.6	matplotlib==3.9.2
pytest==3.8.0	。

步骤 3：获取临机决策部分样例代码。

```
git clone <release_url/text_JSQL.git>

cd text_JSQL
```

步骤 4，配置大模型运行环境，见三、（四）章节

（2）启动

步骤 1：打开 VScode，File-Open folder-text_JSQL 文件夹，打开样例代码，配置 Python 解释器。

步骤 2：打开平台程序，加载既有想定。

步骤 3：运行 Python 案例程序。打开 main.py 文件在 VScode 中点击“Run and Debug”按钮，启动程序运行。（亦可选其他方式打开，例如 Anaconda Prompt 命令行，并运行）

步骤 4：观察平台程序前端可视化态势变化和文本化态势简报，适时介入，以自然语言输入临机决策指令。观察指令执行情况，确认正常生成指令。

（3）创建智能体开发工程（打开 AI Demo）

步骤 1：保持通信类不变，包括 text_JSQL 下的 Env.py 文件等。建议直接继承 text_JSQL 的文件结构。

步骤 2：可重写或修改 agent_guize 目录下 me_AI 文件夹中的小模型部分决策内容，以定制化自己想要的功能。包括 base_agent.py 以及 global_agent.py 中的 Step 函数。应保持相关函数输入输出不变。

步骤 3：可重写或修改面向大模型的提示词以实现更好效果，修改 examples 文件夹中相应的红蓝方 JSON 文件。

（4）调试与运行智能体

步骤 1：启动平台。

步骤 2：确认修改后程序已配置正确的 IP 地址和接口，运行 pytest，确保修改后程序能够通过所有测试。

```
pytest
```

（七）智能体编写规范

SDK 提供了 BaseAgent，用户可基于 BaseAgent 定义自己大模型智能体，示例如下：

```
from jsqlsim.agents.base import BaseAgent
from jsqlsim.llm.prompts import prompts_repo


class ForceComposePlanAgent(BaseAgent):
    def __init__(self, base_url, api_key, model):
        super().__init__(base_url, api_key, model)

    def run(self, budget, target_ships, time_limit, equipment_summary, deployment_constraints):
        prompt = prompts_repo.get("force_compose_prompt_template")

        user_prompt = prompt(
            budget=budget,
            target_ships=target_ships,
            time_limit=time_limit,  # 不需要的字段
            equipment_summary=equipment_summary,
            deployment_constraints=deployment_constraints,
        )

        return self.chat(user_input=user_prompt)
```

四、用户接口设置

（一）大模型 MCP 接口封装

（1）自定义 Prompt

用户可以根据如下方式，使用 prompts_repo.register 装饰器，定义大模型可以使用的提示词，其中参数的描述方式按照 OpenAI 格式，tags 用于后续大模型使用提示词时候，根据 tag 筛选大模型可用的提示词。

```
from jsqlsim.llm.prompts import prompts_repo


@prompts_repo.register(
    tags=["装备编配", "挂载关系"],
    description="根据选定装备和挂载约束生成装备编配 prompt 模板",
    parameters={
        "type": "object",
        "properties": {
            "selected_plan": {
                "type": "string",
                "description": "第一阶段选定的装备规划结果，包含装备名称和数量"
            },
            "mount_constraints": {
                "type": "string",
                "description": "红方挂载约束条件，如平台挂载限制、装备兼容性等"
            }
        },
        "required": ["selected_plan", "mount_constraints"]
    }
)
def compose_equipment_prompt_template(selected_plan, mount_constraints,
```

**args):
    return f"""
你是一名红方作战装备编配专家，任务是将已选定的装备数量，分配成具体的实体单元（挂载关系）。

你需要根据挂载约束，合理安排平台（如机动发射车）与挂载装备（如高成本打击弹），形成可执行的作战实体组合。部分装备可以独立存在（如侦察无人机），不需要挂载。

请严格遵循以下要求：

1. **平台挂载约束必须严格遵守**，如发射车最多挂载几枚打击弹，或者某类弹只能由特定平台携带。
2. 每个平台实体必须明确挂载哪些装备。
3. 可独立存在的装备应作为单独实体列出。
4. 总装备数量不得超出第一阶段结果中的选定装备数量。
5. 输出结果需结构清晰、便于程序处理。

第一阶段规划结果如下：
{selected_plan}

红方挂载约束如下：
{mount_constraints}

请输出结构化 JSON，格式如下：

{{
  "entities": [
    {{
      "type": "平台名称",
      "mounts": {{
        "装备名称 A": 数量,
        "装备名称 B": 数量

```
        }}
    }},
    {{
        "type": "独立装备名称",
        "mounts": {{}}
    }}
  ],
  "reason": "简要说明挂载编配的合理性。"
}}
"""
```

（2）自定义 Tool

用户可以根据如下方式，使用 tools_repo.tool 装饰器，定义大模型可以使用的工具，其中参数的描述方式按照 OpenAI 格式，tags 用于后续大模型使用工具时候，根据 tag 筛选大模型可用的工具。

```
@tools_repo.register(
    parameters={
        "type": "object",
        "properties": {
            "x": {"type": "number", "description": "第一个数字"},
            "y": {"type": "number", "description": "第二个数字"}
        },
        "required": ["x", "y"]
    }
)
def add_numbers(x: float, y: float) -> float:
    """计算两个数字的和"""
    return x + y
```

（二）装备与 UnitType 对应关系

红方装备中文名称到大模型操作的红方仿真装备实体（UnitType）的名称映射如下：

表 14 红方装备与 UnitType 对应关系

装备名称	仿真实体名称	可执行的动作指令
机动发射车	Truck_Ground	发弹（Launch）、机动（Move）、隐蔽（ChangeState）
高成本攻击弹（察打一体）	HighCostAttackMissile	侦察开关（DetectOn\Off）
低成本攻击弹	LowCostAttackMissile	
引导快舰	Guide_Ship_Surface	
侦察无人机	Recon_UAV_FixWing	机动（Move）、侦察开关（DetectOn\Off）

蓝方装备中文名称到大模型操作的蓝方仿真装备实体的名称映射如下：

表 15 蓝方装备与 UnitType 对应关系

装备名称	仿真实体名称	可执行的动作指令
旗舰	Flagship_Surface	发弹（Launch）、机动（Move）、隐蔽（ChangeState）
舰载机	Shipboard_Aircraft_FixWing	发弹（Launch）、机动（Move）、侦察开关（DetectOn\Off）
驱逐舰	Destroyer_Surface	发弹（Launch）、机动（Move）、侦察开关（DetectOn\Off）、干扰开关（JammerOn\Off）
巡洋舰	Cruiser_Surface	发弹（Launch）、机动（Move）、侦察开关（DetectOn\Off）
近程拦截弹	Short_Range_InterceptMissile	/
远程拦截弹	Long_Range_InterceptMissile	/

AIM	AIM_AirMissile	/
JDAM	JDAM_CruiseMissile	/
舰对地巡航导弹	ShipToGround_CruiseMissile	/

（三）大模型战前场景部署接口

AI 端（python）与平台间采用 GRPC 通信，下面是场景部署相关的部分 GRPC 接口.proto 文件：

```
syntax = "proto3";

package RemoteControl;


import "google/protobuf/empty.proto";


service ScenarioEditingService {
    //获取全部装备信息
    rpc getAllEquipageInfo(google.protobuf.Empty) returns (EquipageInfoList){}
    //根据装备名称查找模型参数
    rpc searchModelParameterByEquipage(SearchRequest) returns (ModelParameter){}
    //添加装备实体
    rpc addEquipageEntity(AddEntityRequest) returns (ResponseDate){}
    //更新实体位置
    rpc updateEntityLocation(UpdateLocationRequest) returns (ResponseDate){}
    //更新实体姿态
    rpc updateEntityPosture(UpdatePostureRequest) returns (ResponseDate){}
```

```
    //移除装备实体
    rpc removeEquipageEntity(RemoveRequest) returns (ResponseDate){}
    //更新装备武器挂载
    rpc updateEquipageWeapone(UpdateWeaponeRequest) returns (ResponseDate){}
    //更新装备载荷挂载
    rpc updateEquipageAncillary(UpdateAncillaryRequest) returns (ResponseDate){}
    //更新模型参数（根据装备名称）
    rpc updateModelParameterByEquipage(ModelParameter) returns (ResponseDate){}
    //更新模型参数（根据组件名称）
    rpc updateModelParameterByEquipageModule(ModelParameter) returns (ResponseDate){}
    //获取所有平台列表
    rpc getAllPlatformInfo(google.protobuf.Empty) returns (EquipageModuleList){}
    //获取所有载荷列表
    rpc getAllAncillaryInfo(google.protobuf.Empty) returns (EquipageModuleList){}
    //查找指定平台可挂载的武器配置信息
    rpc searchPlatformWeapon(SearchRequest) returns (PlatformWeapon){}
    //获取全部装备实体
    rpc getAllEntityInfo(google.protobuf.Empty) returns (EntityInfoList){}
    //查找指定装备类型的所有实体
    rpc getEntityInfoByEquipageName(SearchRequest) returns (EntityInfoList){}
    //根据实体名称获取实体信息
```

```
    rpc searchEntityInfoByEntityName(SearchRequest) returns (EntityInfo){}
}
//响应数据
message ResponseDate{
    int32 code = 1;
    string msg = 2;
}

//更新武器挂载
message UpdateWeaponeRequest{
    string name = 1;       //实体名称
    map<string,int32> euipag_weapone_list = 2;//装备武器列表(武器组件名称:数量)
}

//更新载荷列表
message UpdateAncillaryRequest{
    string name = 1;       //实体名称
    repeated string equipage_ancillary_list = 2;   //装备载荷列表
}

//添加实体
message AddEntityRequest{
    string equipage_name =1;       //装备名称(实体类型)
```

```
    string group_name = 2;          //编队名称
    int32 faction = 3;      //所属阵营
    double lon = 4; //经度
    double lat = 5; //纬度
    double alt = 6; //高度
    double heading = 7; //航向
    double pitch = 8;     //俯仰
    double roll = 9;      //翻滚
}

//更新位置
message UpdateLocationRequest{
    string name = 1;      //实体名称
    double lon = 2; //经度
    double lat = 3; //纬度
    double alt = 4; //高度
}

//更新姿态
message UpdatePostureRequest{
    string name = 1;      //实体名称
    double heading = 2; //航向
    double pitch = 3;     //俯仰
    double roll = 4;      //翻滚
```

```
}


//删除请求
message RemoveRequest{
	string name = 1;	//根据名称删除
}


//查询请求
message SearchRequest{
	string search = 1;	//查询名称
}


//装备信息
message EquipageInfo{
	string name = 1;	//装备名称
	string platform_name = 2;//装备平台名称
	int32 faction = 3;	//所属阵营
	string classification_name = 4; //装备分类名称(一级分类：二级分类)
	map<string,int32> euipag_weapone_list = 5;//装备武器列表(武器组件名称:数量)
	repeated string equipage_ancillary_list = 6;	//装备载荷列表
}


message EquipageInfoList{
```

```
    repeated EquipageInfo equipage_info_list = 1;
}
//装备实体信息
message EntityInfo{
    string name = 1;     //实体名称
    string equipage_name =2;       //装备名称
    string group_name = 3;          //编队名称
    int32 faction = 4;       //所属阵营
    double lon = 5; //经度
    double lat = 6; //纬度
    double alt = 7; //高度
    double heading = 8; //航向
    double pitch = 9;     //俯仰
    double roll = 10;       //翻滚
    map<string ,string > parameters =11;//参数名称:参数值
    map<string ,string > attributes = 12;//属性名称:属性值
}

message EntityInfoList{
    repeated EntityInfo entity_info_list = 1;
}

//装备组件
message EquipageModule{
```

```
    string name = 1;     //组件名称
    string classification_name = 2; //分类名称
}

message EquipageModuleList{
    repeated EquipageModule equipage_module_list = 1;
}

//平台武器配置
message PlatformWeapon
{
    string platform_name = 1;       //平台名称
    map<string,int32> weapone_platform_list = 2;//武器平台名称与最大数量列表(平台名称:数量)
}

message ModelParameter
{
    string name = 1;     //名称
    map<string ,string > parameters = 2;//参数名称:参数值
}
```

（1）仿真引擎初始化

首先通过 SimEditor 建立和决胜千里场景编辑器的连接。

```
from jsqlsim import SimEditor

# host 为仿真引擎或者仿真编辑器的服务地址，port 为 grpc 服务端口号
sim_editor = SimeEditor(host="127.0.0.1", port=1333)
```

（2）场景(Senario)

```
# 获取当前引擎默认加载的场景
scenario = sim_editor.default()

# 新建一个默认的场景
scenario = sim_editor.default()

# 加载本地的一个场景 json 文件
scenario = sim_editor.load("test.json")
```

（3）世界(World)

```
world = scenario.world()

# 获取红方阵营
faction = world.get_faction(FactionEnum.RED)
```

（4）阵营(Faction)

当前 JSQL 支持三种阵营，分别为红方、蓝方、未知

```
from jsqlsim import FactionEnum

FactionEnum.RED # 红方阵营
FactionEnum.BLUE # 蓝方阵营
FactionEnum.UNKNOW # 未知阵营
阵营相关接口
from jsqlsim import FactionEnum
```

```
# 获取红方阵营
faction = world.get_faction(FactionEnum.RED)

# 获取红方阵营所有实体列表
entities = faction.all_entities()
for entiti in entities:
    print(entity)
Entity(实体)
创建实体
# 创建实体
entity = Entity(name="无人机", group="", faction=FactionEnum.RED, kind = EntityKind.UAV)

# 设置实体位置
entity.set_location(lon=41.20, lat=20.0, alt=0)

# 设置实体姿态
entity.set_posture(heading=0, pitch=0, roll=0)

# 设置武器
entity.set_weapon(weapon=WeaponEnum.DD, count=10)

# 设置载荷
entity.set_sensor(sensor=SensorEnum.LD, count=1)

# 调用编辑器接口，创建实体
world.spawn(entity)
```

更新实体：

```
entities = faction.all_entities()
for ent in entitis:
    ent.set_weapon(weapon=WeaponEnum.DD, count=100)
    entity.set_posture(heading=0, pitch=0, roll=0)
    # 调用编辑器接口，更新实体信息
    world.update(entity)
```

1）init

定义如下：

```
    def __init__(self, name, group, faction: FactionEnum, kind: EntityKindEnum, lon=None, lat=None, alt=None, heading=None, pitch=None, roll=None):
        self.name = name
```

参数说明 * name: 实体名称 * group: 编队名称 * faction: 所属阵营，详见 FactionEnum * kind: 实体类型，详见 EntityKindEnum * lon: 经度 * lat: 维度 * alt: 高度 * heading: 朝向 * pitch * roll。

2）set_location

设置实体的位置，函数签名定义如下：

```
def set_location(self, lon: Optional[float], lat: Optional[float], alt: Optional[float])
```

参数说明 lon: 经度，lat: 维度，alt: 高度。

3）set_posture

设置实体的姿态，函数签名定义如下：

```
def set_posture(self, heading: Optional[float], pitch: Optional[float], roll: Optional[float])
```

（5） 战前兵力部署相关提示词

```
from jsqlsim import prompts_repo

force_compose: Prompt = prompts_repo.get("force_compose")
result = force_compose(budget=20, target_ships=4, equipment_summary="", deployment_constraints="")
```

```
print(result)
```

1） force_compose

根据预算和目标生成红方作战装备规划 prompt 模板：

```
    tags=["作战筹划", "装备规划"],
    description="根据预算和目标生成红方作战装备规划 prompt 模板",
    parameters={
        "type": "object",
        "properties": {
            "budget": {
                "type": "integer",
                "minimum": 1,
                "description": "预算限制，单位为周"
            },
            "target_ships": {
                "type": "integer",
                "minimum": 1,
                "description": "需要摧毁的蓝方关键战舰数量"
            },
            "equipment_summary": {
                "type": "string",
                "description": "可选装备的汇总信息，包含性能与成本"
            },
            "deployment_constraints": {
                "type": "string",
                "description": "装配与操作约束条件"
            }
        },
        "required": ["budget", "target_ships", "equipment_summary", "deployment_constraints"]
    }
)
```

（2） equipment_compose

根据选定装备和挂载约束生成装备编配 prompt 模板：

```
(
    tags=["装备编配", "挂载关系"],
    description="根据选定装备和挂载约束生成装备编配 prompt 模板",
    parameters={
        "type": "object",
        "properties": {
            "selected_plan": {
                "type": "string",
                "description": "第一阶段选定的装备规划结果，包含装备名称和数量"
            },
            "mount_constraints": {
                "type": "string",
                "description": "红方挂载约束条件，如平台挂载限制、装备兼容性等"
            }
        },
        "required": ["selected_plan", "mount_constraints"]
    }
)
```

（6）战前兵力部署相关工具

大模型可通过如下方式使用对应的 tool

```
from jsqlsim.ll import tools_repo

# 根据名字获工具列表，返回的类表只包含一个元素
tools = tools_repo.get(name="calculate_total_cost")

# 根据 tag 获取对应的 tools 列表
```

```
tools = tools_repo.get(tags=["装备"])
```

本 SDK 提供了如下相关 tool

1） 查询实体详细信息

根据装备名称，获取装备信息。

```
@tools_repo.tool(
    name="equipment_spec",
    tags=["装备"],
    description="根据装备名称，获取装备信息",
    parameters={"type": "object"},
)
```

2） 根据清单计算成本

```
@tools_repo.tool(
    name="calculate_total_cost",
    tags=["装备"],
    description="根据选择的装备清单计算总成本",
    parameters={
        "type": "object",
        "properties": {
            "selected_equipment": {
                "type": "object",
                "description": "装备和数量，如 {\"高成本打击弹\": 2, \"侦察无人机\": 1}",
                "additionalProperties": {"type": "integer"}
            }
        },
        "required": ["selected_equipment"]
    }
)
```

```
def calculate_total_cost(selected_equipment: Dict) -> Dict:
    """计算选定装备的总成本"""
    EQUIPMENT_DB = {
        "无人侦察弹": {"成本": 1, "功能": "用于侦察敌方"},
        "无人干扰弹": {"成本": 2, "功能": "干扰敌方通信"},
        "高成本打击弹": {"成本": 4, "功能": "高精度打击"},
        "低成本打击弹": {"成本": 2, "功能": "范围打击"},
        "机动发射车": {"成本": 3, "功能": "提供发射平台"},
        "侦察无人机": {"成本": 2, "功能": "远程侦察能力"}
    }
    total_cost = 0
    for name, count in selected_equipment.items():
        if name not in EQUIPMENT_DB:
            return f"未知装备: {name}"
        unit_cost = EQUIPMENT_DB[name]["成本"]
        total_cost += unit_cost * count
    return {"总成本": total_cost}
```

（四）大模型战中临机决策接口

（1）Step：向平台发送控制指令。

返回：result 收到的字符串

属性名称	属性类型	是否必填	属性说明
result	string	必填	和以前一样的回传字符串

参数：Action

属性名称	属性类型	是否必填	属性说明
Action	dict	必填	和以前一样的 Action 字典

Action 类型

动作名称（type）	参数	参数种类	说明

Move	Id	string	红蓝方各类装备移动
	lon	float	
	lat	float	
	alt	float	
Change	Id	string	发射车切换状态，枚举类型为机动、隐蔽、起竖。起竖状态无法移动、能被探测，可以开火。
	targetState	enum	
Attack	Id	string	红蓝方各类装备开火。枚举类型为高成本弹、低成本弹、近程拦截弹、远程拦截弹，AIM，JDAM
	weapon_type	enum	
	lon	float	
	lat	float	
	alt	float	
Set_Jammer	Id	string	蓝方舰船打开干扰机
	Pattern	bool	
Set_radar	Id	string	蓝方舰船打开或关闭雷达
	Pattern	bool	

（2）Reset：重置平台状态，开始新对局

（3）get_states：从平台获取当前态势。

返回：statusinfo　收到的字符串

属性名称	属性类型	是否必填	属性说明
redState	dict	必填	红方态势，可解析为字典类型。

blueState	dict	必填	蓝方态势，可解析为字典类型。

（4）GetLandForm：从平台获取当前位置地形。

返回：result　地图属性

属性名称	属性类型	是否必填	属性说明
Landform	string	必填	目标点地形，至少得区分海陆，理想能区分隐蔽区。

参数：LLA

属性名称	属性类型	是否必填	属性说明
Lon	float	必填	坐标经度
Lat	float	必填	坐标纬度
Alt	float	选填	坐标高度

（五）地形查看方式

本平台中地形、坐标等数据可通过 QGis 软件进行查看。注意，这个软件启动较慢，加载地图也不快，请保持适度的耐心。

将文件包中 landform_tif.tif 文件拖入 QGIS 软件。点击下图中位置 1 的图标，之后如 2 所示点击地图中需要知道地貌的位置，在位置 3 中通过 Band 1 查看相关地貌，其中 1 对应陆地，2 对应海洋，在位置 4 中查看该位置的经纬度。

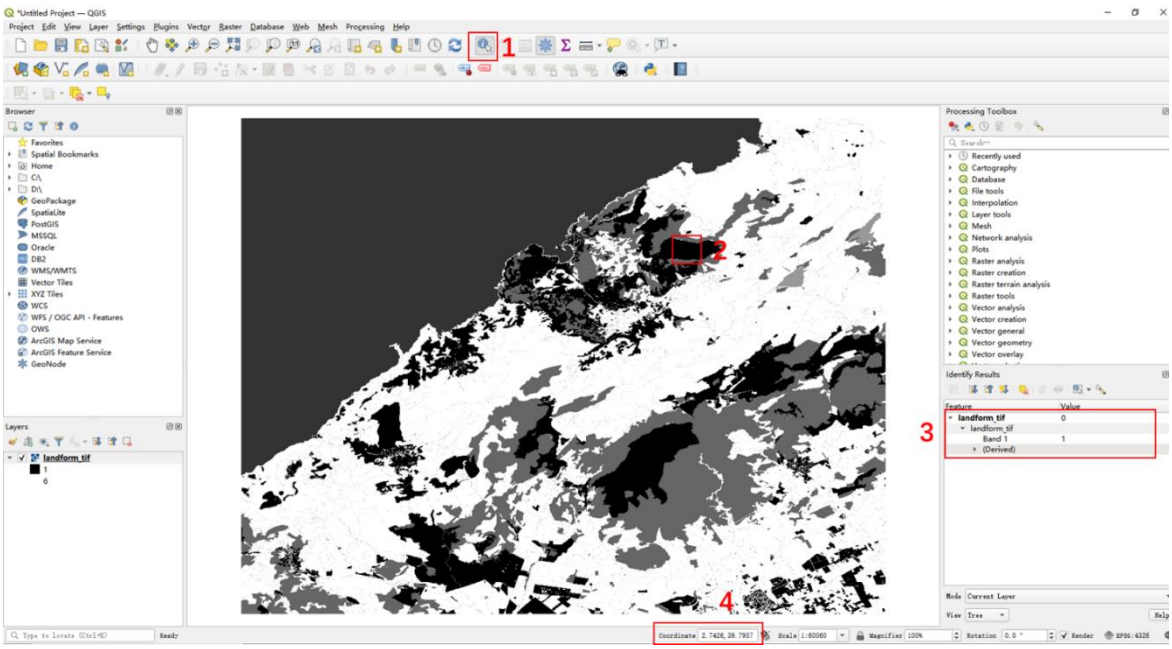

图 5 QGis 界面

此外，直接用 QGis 软件加载地图，可以看到地形地貌，建立直观认识。直接滑动鼠标放大地图，可方便地取出特定位置的精确坐标。

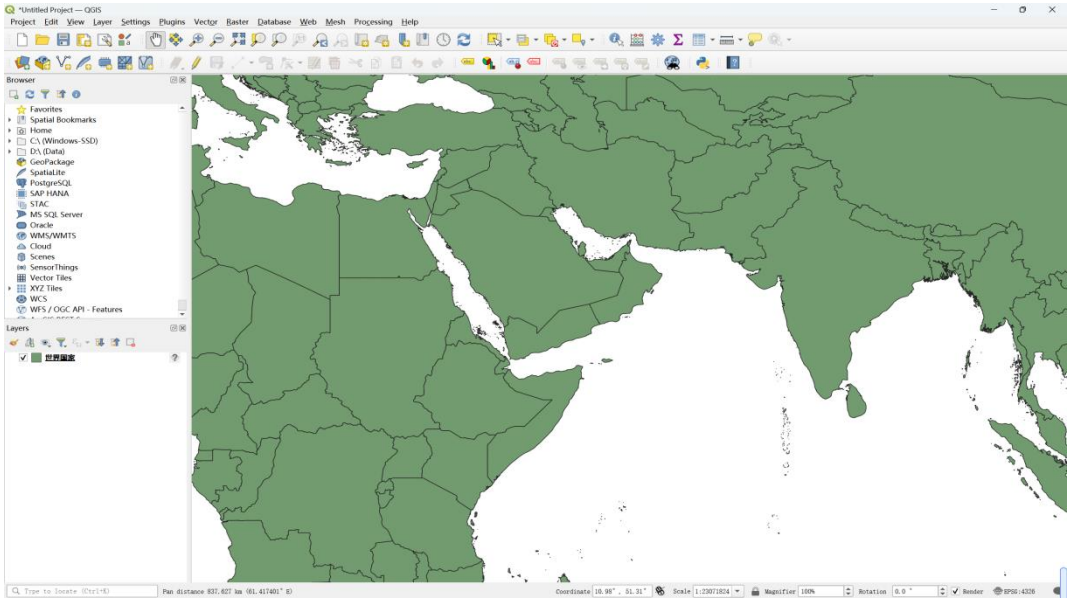

图 6 QGis 直接加载地图