1. Conducting Experiments

    a) How would you do a performance engineering experiment with a reference architecture?

Academic 1	First, I would create a default state for comparison and do a warm-up phase of the system before measuring. Then, you must ensure you are creating realistic load scenarios.
Academic 2	This depends on the goals of the experiment. When using a reference architecture, I would first check for fitting load profiles and then ensure my infrastructure configuration matches my target application. I would also configure, e.g., latency to have a realistic setup.
Academic 3	In a project I did, we defined business events of the application and then defined a tree of failures, stating which sub-failures could lead to the failure of this business event. This is called fault-tree analysis. This helps in systematically defining experiments scoped to specific requirements.
Academic 4	Define the scope and the goal and find tools that assist in reaching the goal. Define SLOs and interesting business scenarios, deploy the system under test, and allocate resources for the load driver. Then, execute the test.
Academic 5	First, define questions and test goals, e.g., how does the test behave under load on black friday. Next, I have to check where I can run the test and how the system needs to be configured. Further, I need test data and need to equip the system with test data. Next, I need to define load profiles, based on historical data. Then, I need to ensure all stakeholders are on board and the boundaries of the test are clear. Now, I can finally run the test and then collect the test data and analyze at to answer my initial question.
Academic 6	First, you need to define what quality attribute you want to test, and you need to differentiate here. Scalability, elasticity, and resilience require entirely different test setups. A tool that supports here should help with a decision tree to come up with fitting test setups. Afterward, the tool should either deploy the system under test or at least execute the generated test and collect the metrics.
Industry 1	In the first place, I would always conduct performance tests for the good case without any failure. This is already done quite seldom in the industry. In general, I would focus on memory exhaustion under load and test once a realistic load and then an extreme load to see when the system breaks.
Industry 2	First, I want to understand the architecture technically and from a user perspective. I need to understand the SLOs for the users and the goals for the experiments. Then, I select proper tooling based on the system's architecture.

b) Would you differ based on the hosting infrastructure or technology stack of the system?

Academic 1	Not too much. A full reset should be done either way to receive a clean state. For comparability, the load scenarios should remain stable in all environments.
Academic 2	Local experiments might be imprecise, but help for getting started and playing around. The results, however, will differ significantly. In general, the execution would be the same, though. Ensure that the system under test and the load driver do not interfere.
Academic 3	Tests always test the entire infrastructure stack. Your results will differ based on the infrastructure tech-stack. So, yes, always think about what you can and have to configure in that infrastructure to create a fitting test.
Academic 4	It should be the same.
Academic 5	Some tests do not make sense locally, otherwise it is the same.
Academic 6	If you can avoid it, you should. If everything is available as IaC, the tool can simply deploy everything based on the given configuration templates. Then, the procedure should be the same.
Industry 1	I would always recommend testing as close to the productive environment as possible. Then, create the tests around that. It depends, however, on how well you can access the application in that target domain, as it might be hidden in a FaaS service.
Industry 2	Not really. What is important, however, is to run a test several times to reduce outliers. Also, ensure that the system and the load driver are not influenced by other systems on the host system.

2. Test Setup (What must one configure to conduct a test?)

    a) How would you set up a performance experiment in general?

Academic 1	First, you need a scalable load-driver. Then, you need a clear runbook - scripted, if possible - of what to test in a reproducible format. You should also check that your hardware is not used otherwise and repeat the experiment several times to reduce side effects. Afterwards, you should protocol everything, from hardware stack to software versions. You should also define an observability level upfront because this itself causes performance overhead.
Academic 2	Create fitting load profiles, configure the system, define the type of the chaos experiment, define the goals of the experiments, and set up monitoring. SLOs in chaos experiments, however, are difficult because you only look at a short period where many things break. This does not necessarily relate to a SLO over a month or year.
Academic 3	For me, a test and an experiment are the same. These experiments can be thought of as a test case, where you have an input an execution and an expected output. Next, you would need a scheduler and an executor to schedule and execute the tests. This includes scheduling and executing the load behaviors and the failure states. Finally, you need to collect the metrics and logs to have an answer to the hypotheses.
Academic 4	Resource allocation, deployment configuration, test definition, monitoring setup, and failure state setup. The configuration is really important to be fixed and versioned, to reproduce the results.
Academic 5	You need infrastructure to run the system, a server that generates load, test data, and observability. Potentially you need to mock some parts of the system and define boundaries of the test.
Academic 6	As mentioned above, a test definition, based on that, a resulting load profile, the system deployment, the test execution, and the collection of metrics and logs.
Industry 1	Your application should be deployed close to production; you need a load driver that is ready to create realistic load scenarios and observability in some kind, which should, however, not cause a performance overhead.
Industry 2	First make some dry runs and play around to get a good understanding of the system and tests. Ensure the reports are in a consistent format. Also, try if you can create load not only at the user interface but also directly on services in the middle.

b) What aspects of the system would you like to configure when conducting performance experiments? (e.g., load generation, failure state)

Academic 1	Load scenarios and generation, failure state, observability level.
Academic 2	Resilience mechanisms such as circuit breakers and retries can be centrally configured sometimes. They must match, otherwise, they can intertwine into issues. Maybe dapr can do this.
Academic 3	Infrastructure, load scenarios, and failure or chaos states.
Academic 4	The same as mentioned above.
Academic 5	Load scenarios, preferably based on real-world data, the chaos experiments and the SLOs which are under test.
Academic 6	Besides the things I already mentioned, failure can also be configured. You can separate configuration things and environmental events. The latter is simulated in the chaos engineering aspect, which should be configurable orthogonal to the load definition. A tool should also segregate those.
Industry 1	Load scenarios and failure states should be configurable, this is already advanced. Also, you should mock surrounding systems.
Industry 2	Configuring the scalability should be possible. Should the system be able to scale based on the load during the test?

c) Should test setup-files be imperative, declarative, or hybrid? Should it be a consistent format or domain-specific?

Academic 1	Depends on the context. It should be ensured not to build yet another wrapper but to use common technology. Also, the level of detail varies based on the experience, it does not make sense to overconfigure everything. A script generator could be helpful, though.
Academic 2	I think creating a proper definition language is difficult. Rather, use something that matches the given tooling, such as Chaos Toolkit. It is challenging, however, to synchronize the failure states and the load states because they are executed by different tools.
Academic 3	This depends on the scope of the project. Declarative is easier for the test definition, but it depends on the scope.
Academic 4	I tried to solve a similar issue with a Kubernetes operator. It takes input files from the SPE domain in a Kubernetes file and maps them to executable steps. The operator then also stores the results in structured folders. Upon that, imperative things can be built. So, maybe a hybrid solution makes sense.
Academic 5	Abstraction always causes information loss. The average developer, who is not to deep into it, however, would benefit from abstraction and especially guidance.
Academic 6	In general, I like declerative languages. However, this makes it difficult to comprehend for users at some points. Being close to the actual tech stack is, however, in order, as it is simpler.
Industry 1	First, setup files are a good idea. Then, it depends on the application you want to test. I have no clear preference here.
Industry 2	Every dynamic test parameter must be reflected in the description to ensure reproducability. An additional descriptive language would be nice, but it should be imperative to clearly describe the test step-by-step. Ensure flexibility by using standards here.

d) Would you like to define a steady-state hypothesis before conducting the experiment? If so, should this be automated? (i.e., collect base state without errors and a default load)

Academic 1	Yes, absolutely.
Academic 2	In theory, you would need to manually define the steady state upfront. In practice, I would use an automated steady-state generation that is approved by the tester.
Academic 3	Measuring and defining it is a good way. Then, compare if both align. Defining a test goal is necessary in any case, though.
Academic 4	This would be a good idea.
Academic 5	This makes sense, since, e.g., in the cloud, you often cannot compare to historical data.
Academic 6	Makes sense.
Industry 1	Good idea, but should be optional as it costs additional time.
Industry 2	Should be done by the tester in the performance case, but if chaos is injected, it makes sense.

e) Would you like test configurations to be stored as code (and possibly versioned)?

Academic 1	Yes, in GitHub.
Academic 2	Yes, absolutely,
Academic 3	Yes, this is also necessary for CI/CD tools.
Academic 4	Yes, this is absolutely required.
Academic 5	Yes.
Academic 6	This would be great.
Industry 1	Absolutely.
Industry 2	Sure.

3. Failure Simulation

    a) What types of failures should be possible to simulate?

Academic 1	Complete failure, connection loss, reduced throughput, latency, increased time of compute inside a service, memory leaks and increasing compute time, maybe recovery scenarios, single components and REST endpoints. Inside the application, the database or web server could crash.
Academic 2	Service crashes. Here, you need to differentiate between service or pod and instances. Infrastructure is difficult to map but could be interesting. Also, delays are useful.
Academic 3	In Kubernetes context, I would mention service failure. Also, injecting delays seems reasonable to me. Further, misconfiguration is a failure state I would like to configure. Wrong dependencies in a container could also be interesting.
Academic 4	I have no particular preference.
Academic 5	Network issues, hardware failure, losing entire availability zones.
Academic 6	Crahes on all layers are helpful. The reason for the crash could be differentiated. Stop faults also should be enough. Byzantine errors are not helpful any further. Any form of delay is also useful.
Industry 1	Cache failure would be great, as it directly impacts the performance.
Industry 2	Network partitioning and latency, service crashes, manipulating the responses.

b) Some examples could be latency, resource exhaustion, service crashes, network partitioning, data integrity failure, and infrastructure issues (e.g., Kubernetes nodes shut down); Pick your top three of those.

Academic 1	Latency, resource exhaustion, network issues.
Academic 2	Service crashes, latency.
Academic 3	Service failure, latency, and configuration issues.
Academic 4	Latency, service crashes, and network partitioning.
Academic 5	Latency, service crashes, and infrastructure issues.
Academic 6	Service crashes, latency.
Industry 1	For me, Latency, resource exhaustion, network issues, service crashes, and infrastructure issues are similar, it's all about a service not being ready. Data integrity is not really testable in performance tests.
Industry 2	Service crashes, latency, infrastructure issues (e.g., configuration issues).

c) At what levels should failures occur? (e.g., service-level (inside the actual software), cluster-level, container-level, network-level, domain-failure)?

Academic 1	Creating failure on service-level provides the most fine-grained and realistic results. However, for simplicity reasons, it should be sufficient for most scenarios only to target the endpoints of the services if there is a proper microservice architecture in use.
Academic 2	Often, you can only work on code-level because in enterprises you might have no access to the infrastructure. This is, however, not very nice. If you have the possibility, cluster-level makes the most sense.
Academic 3	If you ask a practitioner, he would say that every level is interesting. But in your limited time, I would stay on the infrastructure level. This includes the container level, of course.
Academic 4	It depends on the experiment, all could make sense. Personally, I think configurations on the infrastructure level are important to consider. Also, it should be configurable on the replica level and not on the service level.
Academic 5	You can take all of your proposed. And again, it depends on the testing. In the service itself, however, is more complex, so it is probably not the first choice.
Academic 6	Microservice or container level would be the main target point. Then, the infrastructure layer would be the next target level.
Industry 1	Mixing the levels could be difficult, as you can't see what impact individual failures have. Also, I would not change the source code or application container, as this is what is under test.
Industry 2	Start small, infrastructure and container level is probably most simple. If time suffices, of course, check everything.

d) How would you inject failures (e.g., code-level, middleware, infrastructure-level, sidecar, in the container)?

Academic 1	On code-level, if you want to be precise, on service-level, you can choose sidecar or in the container, but the closer to the actual application, the better.
Academic 2	As mentioned, a sidecar or in the container would be fine from a research perspective.
Academic 3	Sidecars make sense, as MiSArch already uses sidecars. Make sure it targets important domain points in MiSArch to create useful tests.
Academic 4	As mentioned, on replica level, so for each container individually.
Academic 5	Using the infrastructure level mainly, I would count a sidecar there, too.
Academic 6	I like the use of sidecars.
Industry 1	Infrastructure-level and sidecar would be best; don't touch the applications.
Industry 2	Using a sidecar seems to be reasonable. Of course, on code-level would be nice, too. But also check infrastructure failures. Those should be pretty simple to set up, too.

e) Should failure be static, dynamically changing during the test, or both?

Academic 1	Dynamic enables more realistic results, but it depends on the complexity you want to test. Start with a simple approach and ensure it works.
Academic 2	First, focus on static failure, then check if dynamic failure is possible.
Academic 3	Many tests require dynamic behavior. It depends on the test that is executed. Both should be configurable.
Academic 4	I have no strong opinion, but dynamic failure seems to be more interesting from a resilience perspective.
Academic 5	Both is important, there are many different scenarios that are thinkable.
Academic 6	I like dynamic failures, especially since this is not easily covered with models and simulation.
Industry 1	Dynamic is a good possibility, but you should always start with static values.
Industry 2	Both.

f) Should failure be only deterministic or also non-deterministic?

Academic 1	For reasons of reproducibility, deterministic should be the first choice, but if you want to test against overfitting and the resiliency of your system, non-deterministic failures could be interesting, too. Of course, it needs to be configurable and choosable.
Academic 2	Rather deterministic on a reference architecture, to define clear experiments. However, if it is not a huge overhead, go ahead and enable non-deterministic failures.
Academic 3	Both should be possible.
Academic 4	Both should be possible.
Academic 5	I think, deterministic seems to provide more results, if you define a hypothesis upfront.
Academic 6	I would start with deterministic. This can still be extended in the future. Especially since the analysis and interpretation are more difficult.
Industry 1	It should always be reproducible, in my opinion.
Industry 2	Both are good, depending on the test case.

4. Load-Testing Configuration

    a) How should load tests in experiments be configurable? (e.g., request rates, concurrent users, traffic patterns, duration, target interfaces, fuzzing)

Academic 1	Use real-world load data from the domain and think of personas. A normal user persona is necessary, but a brute-force attack could also be interesting. Make sure to model think-time between requests. Load patterns should be scalable by configuration.
Academic 2	Check if you want to use open or closed workloads. Not every tool can do everything. Open workloads are more realistic, in general. Further, the examples you mentioned make sense.
Academic 3	The examples you provided are enough.
Academic 4	User behavior is not explicitly mentioned. This is something I would add to your examples. Check on Gatling, as it has a nice DSL for such definitions.
Academic 5	Work, i.e., at least HTTP endpoints needs to be configurable, and sessions need to be possible. Model think-times and make sure to have a distribution here, to reduce synchronization effects. The amount of starting sessions per second (open load profile) is my preferred load configuration parameter. Ramp-ups and cool-downs are necessary, too. Dynamic load patterns would be great.
Academic 6	You have to differentiate between work and load. The test generation wizard should generate load patterns based on the input questions. Work, however, is hard to automate as it is domain-specific. Here, your tool must offer the possibility to input fitting work, and this must be provided by the user. The load parameters you mentioned fit.
Industry 1	Request rates, concurrent users, traffic patterns, duration, target interfaces. Fuzzing should be done only reproducible. Make sure it matches the domain.
Industry 2	Fuzzing is a must-have. Otherwise, you might always run in the same cache, which you want to circumvent. Then all the things you mentioned should be there.

b) Should there be predefined workload test scenarios?

Academic 1	Yes, it helps to get a quick understanding. This could then be extended by requests that can be scraped from the API definition of the system. Such a pre-defined scenario could be defined in JMeter or k6 and deployed in a Docker container.
Academic 2	Yes, this makes total sense.
Academic 3	Yeah, this would be very interesting.
Academic 4	Yes, totally.
Academic 5	Yeah, as a guidance for the tool itself. For average users, this also helps for a general understanding.
Academic 6	Yes.
Industry 1	Yes, give some simple examples for MiSArch and, in general, how to use the framework.
Industry 2	Yeah, it would help for an easy knowledge transfer for the reference architecture.

c) Would you like real-world traffic replication features? (i.e., some predefined load profiles with realistic values and scenarios)

Academic 1	Yes, as already mentioned, use it from the domain.
Academic 2	Yes, if you find available profiles online, provide them for the tool.
Academic 3	This would also be a cool feature. There is data available about this online, e.g., for the football world cup tickets
Academic 4	Yes.
Academic 5	If you have real-world traffic, use it. Load tests are very complex, so having a realistic picture, in my opinion, is good enough in practice.
Academic 6	If this is possible, this would be cool.
Industry 1	Yes, but also give some rationales on different other domains that show what realistic load patterns look like.
Industry 2	Yes, it is useful, but ensure to test beyond expected load profiles.

5. Experiment Execution

   a) How should experiments be started? (e.g., via API, CLI, UI, CI/CD pipeline integration)

Academic 1	CI/CD integration is a nice idea but not really necessary for the thesis. A CLI and API integration should suffice.
Academic 2	CI/CD integration is a nice idea, but focus on the local usability right now, especially since you are using a reference architecture. An API is the most flexible solution.
Academic 3	Many of the modern tools have all these options. In terms of priority a clean API would be the highest, then a CI/CD integration.
Academic 4	All those methods serve different use cases. A simple job definition injected in the cluster should suffice when using a Kubernetes operator. This can be launched via a CLI. UIs are always hard to map properly. CI/CD pipelines are completely unnecessary for me.
Academic 5	API and CLIs are usually executable in pipelines, even though I don't believe complex experiments are used in many pipelines. However, for non-experts UIs are very helpful, so I would appreciate a UI for the test execution and generation.
Academic 6	Focus on the CLI, as it uses an API internally, anyhow.
Industry 1	A GUI is nice to define the test, intuitively. An API and CLI are then good for executing the test. CI/CD integration would be nice, but if there is a CLI integration, this is straightforward.
Industry 2	A CI/CD integration is necessary for enterprises to run the tests on new releases regularly. A user interface for business users and creating test scenarios can help, too, of course.

b) Should experiments be scheduled or triggered by specific conditions?

Academic 1	Obsolete if no CI/CD integration is done.
Academic 2	Obsolete if no CI/CD integration is done.
Academic 3	By any CI/CD-pipeline trigger.
Academic 4	Obsolete if no CI/CD integration is done.
Academic 5	A nightly pipeline could be interesting.
Academic 6	Obsolete if no CI/CD integration is done.
Industry 1	Whatever is offered by the CI/CD pipeline.
Industry 2	By new releases in the CI/CD pipeline.

6. Test Results and Visualization

   a) How should test results be presented? (e.g., dashboards, logs, reports)

Academic 1	Metrics are often domain-specific and not reusable. A simple dashboard that uses some basic traces and metrics, e.g., with Grafana and Jaeger, is nice. However, it is important to store all the observability data in a dedicated directory with a timestamp.
Academic 2	A dashboard makes sense. Use something that exists, like Grafana. Reports and logs are not read thoroughly in practice.
Academic 3	Use a common format such as Prometheus/Grafana and provide dashboards.
Academic 4	Yes, dashboards would be nice. However, custom reports and metrics are only applicable in MiSArch. When switching the system under test, those might not be available. Choose general metrics.
Academic 5	A textual, versionable document (HTML, markdown) helps, because I simply can save it and take a look at it later. Dashboards, are great, too, to conduct complex analyses and investigate the metrics. A customer would always want a report as well.
Academic 6	A multi-modal report that contains visuals, such as graphs, textual explanations, and logs or traces where it makes sense would be cool. Usually, this would be LLM-generated. However, start simple with some basic and common dashboards such as Grafana.
Industry 1	First, a report from the user perspective is key. This should be visualized over time. But also the service metrics (CPU/memory, response times, and error rates) need to be visualized over time. Make sure the outliers are highlighted.
Industry 2	Correlate logs and traces with timestamps by injecting additional logs for the services that match the test execution. If this is to complex, at least give a guideline on what to log and how to use OpenTelemetry. Further, the tool should generate a textual report that is machine-readable. Maybe there is already standard tooling.

b) Which key metrics are most important for assessing performance and failures? (e.g., response times, throughput, error rates, CPU/memory usage)

Academic 1	Response times, throughput, error rates, CPU/memory usage on different levels and services. But don't over-engineer; take what you got.
Academic 2	Response times, throughput, error rates, CPU/memory make total sense. Also. measure infrastructure metrics like the number of active pods and instances. There might be a diff between what the auto-scaler measures and what is there.
Academic 3	The content of the messages could also be interesting, but the ones you mentioned should suffice.
Academic 4	Application level metrics, like the number of requests, messages in the message queue, etc., would also make sense.
Academic 5	The ones you mentioned suffice.
Academic 6	The number of pods or replicas over time is also interesting.
Industry 1	Exactly those.
Industry 2	Those are good, but measure all services. Make sure to properly measure the correct memory parts, e.g. Heap, native.

c) Should historical experiment results be stored and compared?

Academic 1	Yes, comparability is key for reproducability, but this does not need to be automated in the report.
Academic 2	Yeah, this could make sense.
Academic 3	That is a cool idea. In the Kano model, I would rank this as an excitement feature.
Academic 4	Yes.
Academic 5	In practice people want to know this. Ensure, that you only highlight differences if they are significant enough, as you always have deviations.
Academic 6	Yes, it makes sense. Maybe you need to filter the data upfront and not store every data point since large amounts of data are created in a test.
Industry 1	Yes, it would be helpful, but it does not need to be in a dashboard.
Industry 2	Yes, mandatory.

d) Would you like automated analysis and recommendations based on test results?

Academic 1	Would be nice to have, but is not a priority.
Academic 2	I don't know if it works out of the box, but it would be nice to test it and see the results.
Academic 3	Certainly, this is a cool idea and another excitement feature.
Academic 4	That would be nice. The analysis is the most difficult part. Getting support from an AI for this is a great use case.
Academic 5	This idea has great potential, so, yes.
Academic 6	The most time-consuming part of such a test is the analysis. Having an LLM support you here would be extremely helpful.
Industry 1	This would be great, of course, if you send all data and the historic test results.
Industry 2	Yeah, this would make analyzing so much more insightful, and one could learn from this.

7. Relevant Metrics for Meaningful Results

    a) On what level should metrics be gained (e.g., application level, container level, host level)?

Academic 1	On all levels, at least some log is necessary. Store everything you can get.
Academic 2	On all available levels, but especially container and infrastructure level.
Academic 3	On all available levels.
Academic 4	On all levels.
Academic 5	On all levels, as well as the load driver.
Academic 6	On all levels.
Industry 1	On container- and infrastructure layer. Service-Level might be cumbersome due to proprietary frameworks. In general, collect as much as possible.
Industry 2	On all levels if possible, definitely on all containers and the infrastructure.

b) Are there SLO thresholds that should be considered? (minimum and average)

Academic 1	It could be helpful in the reporting to see if some SLOs are not met, but for the beginning, keep it simple and just look at the data.
Academic 2	As said, SLOs are difficult in the context of chaos experiments. Rather, define a goal that matches your test.
Academic 3	You would define a structure for the experiment, where the user can inject his goals, and then this gets evaluated.
Academic 4	This must be defined in the experiment definition.
Academic 5	Yes. I would calculate the percentage of requests that does not meet the threshold, then look if it meets my SLO.
Academic 6	Yes, this would make sense, but it should be the goal of the test not a generic SLO.
Industry 1	Usually, you have some SLOs upfront. It could be nice to directly compare them, but it's not a high priority.
Industry 2	Yes, this should be possible.

c) Should performance impact be analyzed at different levels (e.g., user experience, system infrastructure)?

Academic 1	Yes, from a load-driver perspective, but also a service and host level perspective it is necessary to analyze the impact.
Academic 2	Depends on the user of the dashboard. Maybe you need two different views, one for the user/tester and one for the engineer.
Academic 3	Yes, from the user and system perspective.
Academic 4	Yes, from the user and system perspective.
Academic 5	Yes, from the load driver and system perspective.
Academic 6	Yes, this probably makes sense.
Industry 1	Yes, from the user and system perspective.
Industry 2	Yes, from the user and system perspective

8. Automation Preferences

    a) How much automation should be incorporated into the performance engineering process?

Academic 1	You can look at MiSim. They designed a YAML schema that describes failure and workload, and then their simulator parses this input and creates the simulation out of it. Something like that is a good idea.
Academic 2	Test execution, failure injection, and metrics collection should be automated. Defining test scenarios could be supported by the tool but not automated.
Academic 3	Everything possible should be automated from the execution. The test generation should be tool-supported, maybe also by using generative AI.
Academic 4	The test creation is hard to automate. Everything from execution to failure injection and metric collection must be fully automated. You should create a management layer for existing tools, that, if necessary, you can also target fine-grained adaptions, e.g., in the Gatling DSL.
Academic 5	The test execution and collection of metrics and report generation sohuld be entirely automated. Supporting the test execution in the UI is also super helpful.
Academic 6	Creation of the test setup file using the wizard, and then automated execution, failure injection, metric and log collection and reporting using an LLM.
Industry 1	The execution, of course, should be automated. The test setup could be supported by a Wizard, but fine-tuning will probably be done in the generated test config file.
Industry 2	Ensure automation of the entire execution and reporting is in place so you can regularly run the test as an integration test in your deployment pipeline and compare the results to historic baselines. For the start, a generating of load profiles also is helpful.

b) What aspects should be automated? (e.g., load test generation, failure injection, test execution, result analysis, reporting)

Academic 1	Load test generation, failure injection, and test execution based on the input is good. A simple dashboard with results is also nice, but for everything else, manual work with the logs is necessary.
Academic 2	As mentioned above.
Academic 3	As mentioned above. The analysis of the results could then be done by a human or supported by an LLM.
Academic 4	As mentioned above.
Academic 5	As mentioned above.
Academic 6	As mentioned above.
Industry 1	Failure injection should also be automated based on the test input, and an automated dashboard with the results would be good. Load execution should be done, too, as already mentioned.
Industry 2	The entire execution and reporting.

9. Non-Functional Considerations

    a) What is your most important non-functional consideration for a performance engineering framework?

Academic 1	Reproducibility. On the one hand, others should be able to redo your experiments, but it should also be possible to execute them on other architectures. Thus, logging quality is important. Every change that the framework does in terms of load, failure, etc., needs to be documented well with timestamps.
Academic 2	No performance impact and reproducability. The load driver should not influence the measured output, it must be a different system.
Academic 3	The timing aspect is important. The experiment must end at some point, and generating the report should not take long. Also, efficient resource usage is important. Finally, interoperability with other tools is important.
Academic 4	Performance isolation is important. The performance of the load test must not influence the performance of the system under test. This can be done with a dedicated node in Kubernetes. Also, it must be usable and reproducible.
Academic 5	Efficiency in load generation. The usability in terms of operability should be good, the tool short work out-of-the-box.
Academic 6	Usability. Also, in terms of self-explainability. Provide helping texts and guidance for inputs. For example, how is the load generated - open or closed?
Industry 1	The setup should be lightweight and easy to use. The reports should be generated quite fast after each execution. Reliability is also important, the tool should not randomly crash.
Industry 2	It should be resource-efficient, robust, and transparent if there are any constraints. Usability, of course, is important, too.

b) Would you use the chaos-based performance engineering tool in other domains?

Academic 1	Yes. Therefore, it should be rather generic and use common tooling.
Academic 2	Makes sense because tech stacks differ based on the use case.
Academic 3	Yes, other domains should also be possible.
Academic 4	Yes.
Academic 5	Yes, absolutely.
Academic 6	In general, yes. But as you know, the different tech stacks always differ. Try to focus on MiSArch in the first place, but keep flexibility in mind when designing the architecture.
Industry 1	Yes, absolutely.
Industry 2	Yes, totally.

10. Additional Considerations

    a) Are there any industry best practices or frameworks you prefer for performance engineering?

Academic 1	Create a super basic mini example subset of the framework that shows how it works with a simple container to get get testers started quickly. Measure monitoring overhead in performance and make it transparent.
Academic 2	I already mentioned them in the above answers. For tooling, I would use Chaos Toolkit and maybe Gatling.
Academic 3	-
Academic 4	As mentioned, Gatling for the load creation and my Kubernetes-operator for the test management and execution.
Academic 5	Gatling, especially due to the open workload model.
Academic 6	-
Industry 1	-
Industry 2	Convert the results in a reusable format that actually creates business value, e.g., Excel. And test as much as possible.

b) What challenges have you faced with performance testing in microservices?

Academic 1	There are different load behaviors, open and closed. Some load-testing tools tend to reduce the load if there are errors, thus, the results are imprecise. Also, single-core microservices without concurrency can cause massive bottlenecks.
Academic 2	-
Academic 3	-
Academic 4	-
Academic 5	Highly interwoven systems are really hard to separate and to test individually.
Academic 6	The tech stack is the largest issue most of the time. Also, the infrastructure and hosting OS always differ and then cause several issues. Thus, focus on MiSArch as this is fixed, but keep flexibility in mind.
Industry 1	You must check that your infrastructure allows monitoring and failure injection.
Industry 2	The tool must be scalable to ensure it handles large systems under tests.

c) Are there any specific compliance or security considerations for performance testing in your organization?

Academic 1	-
Academic 2	-
Academic 3	-
Academic 4	-
Academic 5	Security and compliance with regards to user data. Of course you want to use real data, if possible, but you must ensure it is not leaked anywhere.
Academic 6	-
Industry 1	Never load test in production, but always really close to production.
Industry 2	Check the interfaces for the system under test and ensure you do not overly stress other systems or mock them if necessary. Ensure that you do not test on production. Check that you don't expose user data somewhere.

d) Is there anything else you would like to add?

Academic 1	Look at Armin van Hoorn's work and his research group.
Academic 2	I have conducted a project with domain storytelling. This is a super basic definition for business professionals, which allows the creation and execution of chaos experiments on specific points in the business process. This could also make sense for you.
Academic 3	-
Academic 4	-
Academic 5	Take a look at BenchFlow.
Academic 6	-
Industry 1	-
Industry 2	-