Global Under-Resourced MEedia Translation (GoURMET)

H2020 Research and Innovation Action
Number: 825299
D5.3 – Initial Integration Report

Nature	Report	**Work Package**	WP5
Due Date	30/06/2019	**Submission Date**	30/06/2019
Main Authors	Andrew Secker (BBC), Susie Coleman (BBC), Mikel L. Forcada (UA), Anna Blaziak (BBC), Rachel Bawden (UEDIN), Radina Dobreva (UEDIN), Felipe Sánchez-Martínez (UA), Víctor M. Sánchez-Cartagena (UA)		
Co-authors			
Reviewers	Barry Haddow (UEDIN)		
Keywords	translation, API, system architecture, training		
Version Control			
v0.1	**Status**	Draft	20/6/2020
v1.0	**Status**	Final	29/6/2020

Contents

List of Figures

1 Introduction

This document describes how the translation models, supplied by research partners, have been integrated and are made available via a cloud-based translation service.

A single platform has been created in which GoURMET translation models can be hosted, run and accessed in a standard manner. With the correct provision around security and mediation of access by 3rd parties this single platform can support numerous prototype tools across multiple project partners.

The advantages of locating the translation technology in one place and then mediating access to it via a service is this provides a single point for maintenance and updates. In contrast, if each prototype (tool, experience, etc.) has the translation technology integrated directly, an update to the translation technology (whether that be an improvement to the translation models themselves, a bug fix, a security improvement, etc.) must be undertaken numerous times.

There are, however, a number of considerations for a single translation platform that would not be present if the translation technology were built into each individual prototype or tool.

- Since the platform now provides a single point of access, the platform must scale in response to the number of requests made from a variable number of tools and their users. Scaling must be automatic and reactive based on incoming request load.
- With the platform now representing a single point of failure, it must be robust with automatic detection of failure and the ability to seamlessly move the servicing of incoming requests away from a failed process in a manner such that the end user sees no break in service.
- As the translation service is accessed by multiple users, access must be restricted to authorised parties and thus parties must identify themselves when submitting requests. It is also prudent to ensure that no one party can overload the platform with requests. While the service is required to scale, resources are not infinite and as such it is reasonable for the service to gracefully decline an unreasonable rate of requests from a single party in order to maintain a reasonable level of service for others.
- It must be straightforward to apply updates to the service and these should not result in a noticeable break in responsiveness from the user's perspective.
- External input and output to the translation modules occurs via a well defined API.

The remainder of this document describes the technical details of how the service was built.

The following subsection (Section 1.2) provides a conceptual overview of the service to be built. Section 2 follows by describing the agreed method by which the language models are packaged and delivered to the integrators. Section 3 describes the architecture of the system which makes those models available as a service and considers issues such as security and scalability of the platform. Section 4 details the API which ultimately makes the translation service usable. Finally, a basic browser-based demonstrator UI is presented in Section 5.

The scope of this document has been expanded from that originally planned. Appendixes A and **??** record the training and validation of the translation models that were integrated. These appendixes therefore act as a bridge to Deliverable D5.4 (Secker et al. (2020)), which details the evaluation of the deployed models.

1.1 Workpackage 5 Context

The aim of GoURMET WP5 is to take the output of the research partners in the form of translation models, make them available to a media partners' production environment for the production of prototype tools and experiences, and evaluate the quality of the translation models.

This deliverable is an interim report on Task T5.2 from the GoURMET description of work.

For reference, we include the description of T5.2 from the Grant Agreement:

Task T5.2: Creation of shared interfaces

> *This task is split into three parts: the creation of a managed "production" platform; the rapid prototyping of a shared web-based user interface; and the realisation of an API that can be used by custom integrations. Both the web-based user interface and API will allow access to all translation modules as they become ready.*
>
> *A platform will be built which will contain the research technology generated by previous workpackages. This will be run as a "production" system, distinguishing it from any "development' implementations run by project partners as part of their own research or development processes. Project partners shall provide their contributions in the form of docker modules and implementing an agreed interface standard. Learnings from previous projects suggest strongly that a managed platform such as this will promote version control, platform availability and platform stability. This platform shall be cloud-based and deployed on a reputable service such as AWS. Management of this platform will be ongoing for the remaining duration of the project.*
>
> *Based on the requirements gathered in [Task T5.1, Requirements Gathering, See GoURMET Deliverable D5.2 Secker et al. (2019)], development planning is started and technical partners will add non-functional requirements, setting system related expectations and performance acceptance criteria, explaining how – and how well - the system should work. The focus will be on the rapid creation of a shared, web-based prototype centrally located and managed. The creation of such an interface will allow the BBC and DW to expose the research to users at a very early stage, and thus enabling feedback to be given by the industrial partners to the research partners from an extremely early stage of the project. This ongoing feedback will be a key theme throughout the rest of the project and will be enabled greatly by a user interface shared amongst all partners.*
>
> *An API must then be defined in order to provide an interface between the research technology generated by previous work packages and the custom integrations realised by the BBC and Deutsche Welle in [Task T5.3, not covered in this document, see Section 7]. The API shall use the principles and good practices of RESTful interfaces. Security of the interface shall be a consideration from the outset with HTTPS used for communication and certificate-based authentication strongly desired.*

1.2 Overview of Integration Work

The integration activities in WP5 cover a number of processes. A conceptual summary of the overall WP5 integration activities can be seen in Figure 1. Ultimately, the primary purpose of these activities is to take the output of the research partners indicated in red on the right-hand side of the diagram, and make those translation models available to the generic integrations on the left hand side using a translation service.

Figure 1 shows a basic overview of how this is achieved. Each integration point is marked with a reference to the appropriate integration specification. Integration specifications, documents Coleman et al. (2020b) and Coleman and Secker (2020) are internally published specifications and available as companion documents to this report on request.

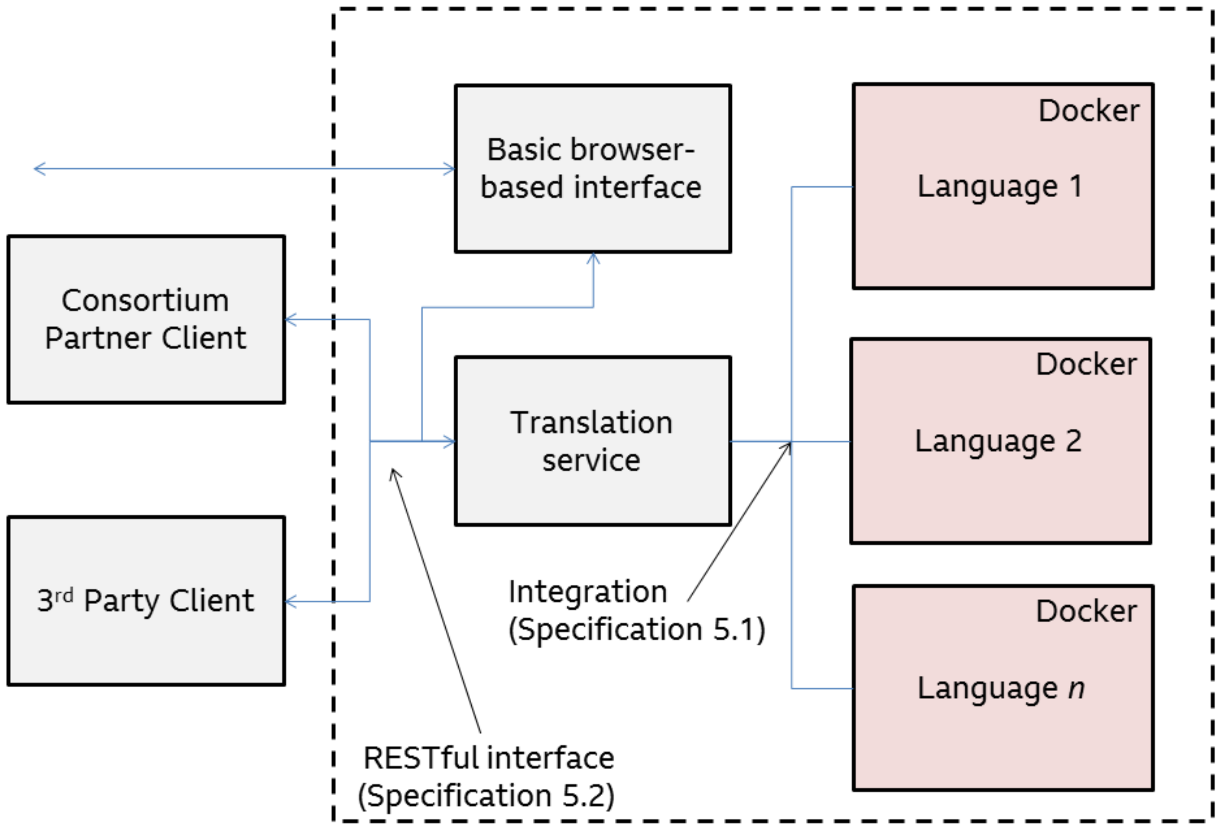

Figure 1: Conceptual summary of WP5 integration activities

The dotted box in Figure 1 indicates the scope of this document. Descriptions of consortium partner clients indicated are within the scope of GoURMET WP5 and development is underway. These will be covered in future deliverables.

2 Translation Model Delivery and Integration

The translation models described in the appendices A and **??** are delivered to the integrators according to an agreed specification in order that translation can be made available as a service, and therefore used by the wider project partners and, possibly, externally.

The requirements around the delivery of the translation modules[1] allow the integrators to treat the output of the research partners as a black box. This has the advantage of allowing the research partners to choose almost every aspect of how the translation takes place. i.e. NMT framework, programming language used, testing, updates, and so on. The advantage to the integrators is that only one integration template need be produced to support any language pair from any research partner.

Figure 2 provides a high level overview of the language model integration. *Integration points* are arrowed in this diagram. At each of these integration points the system must conform to a specification in order to allow interoperability and compatibility between technologies.

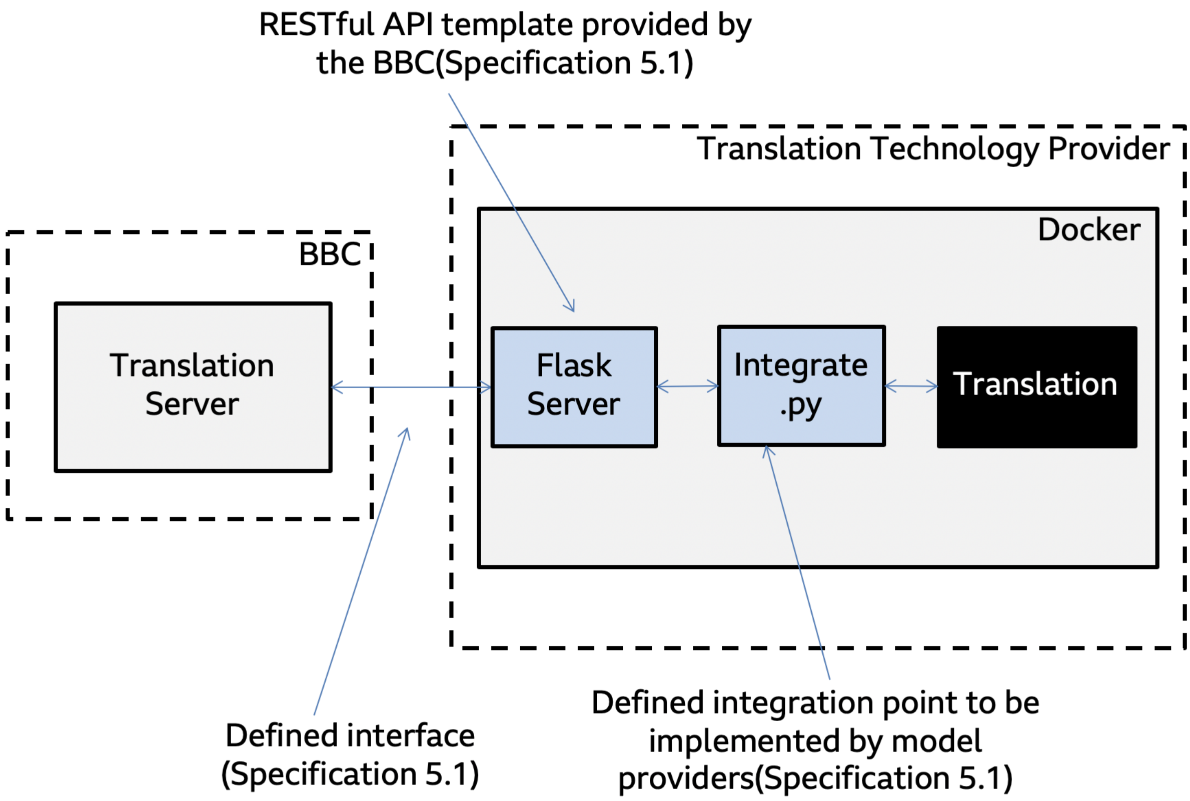

Figure 2: Block diagram of integration architecture for translation module

Specification 5.1 -Translation module and version control specification Coleman et al. (2020b) is an internal specification document published by GoURMET Workpackage 5 and is available as a

[1] The phrase ”translation module” is deliberately used within this section so as to be non-specific about the nature of the technology performing the translation operation.

companion document.

The general principles around translation module delivery are:

- Each translation module is delivered to the integrators as a prebuilt Docker image
- The Docker image contains a web-server (built using Flask HTTP server framework) which will expose a simple RESTful service. This service will
  - provide the integrators with a sole integration point for the translation module and
  - be built and maintained by the systems integrators guaranteeing robustness and future compatibility with any integration
- Translation technology providers need only customise a single Python file `integrate.py` in order to connect the Flask HTTP server to their specific translation module.

Each Docker image supports one language pair, translating in one direction.

The architecture above has been specifically chosen to support the general translation service architecture (Section 4). The RESTful API approach allows for a long life cycle for each Docker container as once started the container will continue to run until manually stopped enabling a single container to service multiple requests. As a result any (potentially time consuming) initialisation only has to happen once and not before every translation. Multiple Docker containers may be started, run simultaneously and terminated at will, depending on system load (frequency of incoming requests).

It is important to note that the RESTful service exposed by the Docker container is only applicable for the integration with the Docker container, and is not suitable for general or wider integration or exposure for 3rd parties. The security, scalability, etc. required to support such interaction is the task of the translation service infrastructure as described in Section 4.

2.1 Building a Compliant Docker Image

The building of a Docker image for delivery to the integrators is a two-stage process, using templates at each stage to ensure compatibility.

First, an application compliant with Coleman et al. (2020b) is built using the template provided by the systems integrators (published for the project partners on github). This template provides:

1. A standard directory structure for the container. Directories and files can be added but not removed.
2. A basic RESTful API using the Flask web framework
3. A template `integrate.py` file

The translation technology provider can add to the translation module(s) any ancillary scripts, libraries, training/texting data or anything else that is required. In addition to this the translation technology provider must also amend the template `integrate.py` (see Section 2.2), in order to connect up their translation module with the provided Flask server.

Once the application is complete and tested, a compliant Docker image is built using the Dockerfile template provided by the systems integrators (published to the project partners on github).

2.2 Integrate.py

`integrate.py` is a key piece of the integration as it connects the Flask server to the translation technology. Translation technology providers are required to implement a small number of defined methods in the `integrate.py` script such that the correct response is obtained from the translation module when the method is invoked.

The two methods that must be completed by the translation technology provider are:

`init()` The purpose of this function is to perform any set up required before translation can begin. It is also an opportunity to optimise performance, for example allowing the model to be initialised when the Docker container starts instead of each time a translation is performed. This function will be run exactly once, typically during initialisation but guaranteed to be before the first call to the the `translate` function is made.

`translate()` Takes a string encoded as UTF–8 characters in the source language and returns a UTF–8 encoded string representing the translation in the target language. This is called every time a translation is required

The above is simplified for the sake of clarity. Full method signatures, error handling and logging requirements are specified in Coleman et al. (2020b)

2.3 Integration API

Not to be confused with the externally facing API, the Flask server embedded in the Docker image, provides the sole means of interacting with the translation models at runtime. As a result this Integration API service is the sole means by which the main translation service (described in Section 4) can request translations from the underlying MT models.

The integration service supports only one method of interest:

/translate (HTTP POST) Translates from the source language to the target language. No language and direction need be specified as each Docker image translates only one language pair in one direction. Takes UTF-8 encoded JSON data in POST body in the form: `"q"=string` where `string` is the text to translate.

Input checking, error handing, status querying, etc. on this interface is deliberately minimal. This service can assume any input has been checked and sanitised, as the systems integrators are in control of that step. Likewise, the requesting service can assume that all input will be successfully translated.

The only system to interact with this integration API at runtime is the translation service, as described in the following section.

3 Translation Service System Architecture

The cloud-based translation service forms the core of the WP5 integration activities. In order to make the translation models available for use in prototype tools, third party integrations and so on, a single platform has been created in which the translation models can be hosted, run and accessed.

To define some terms, the *platform* is designed to support numerous heterogeneous prototype tools across multiple project partners by offering translation as a *service*.

The advantages of locating the translation technology in one place and then mediating access to the technology via a service are numerous. This deployment paradigm provides a single point for maintenance and updates. In contrast, if each prototype, tool, experience, etc. has the translation technology integrated, an update to the translation technology (whether that be an improvement to the translation models themselves, a bug fix, a security improvement, etc.) must be undertaken numerous times across numerous installations and quickly becomes unmanageable.

However, there are a number of requirements for a translation platform that would not be present, or present in a different form, if the translation technology were built into each individual prototype or tool. These are key considerations and have been taken into account when designing the overall system architecture described in this section.

- **Longevity.** Ensuring the architecture remain the right choice as the project scales and matures

- **Re-usability.** Leveraging existing solutions to save time and cost if possible, rather than building from scratch

- **Cost.** Both initial setup costs and how cost scales with demand

- **Scalability.** Since the platform now provides a single point of access, the platform must scale in response to the number of requests made from a variable number of tools and their users. Scaling must be automatic and reactive based on incoming request load.

- **Resilience.** With the platform now representing a single point of failure, it must be robust with automatic detection of failure and the ability to seamlessly move the servicing of incoming requests away from a failed process in a manner such that the end user sees no break in service.

- **Security.** As the translation service is accessed by multiple users, access must be secured to authorised parties and thus parties must identify themselves when submitting requests. It is also prudent to ensure that no one party can overload the platform with requests. While the service is required to scale, resources are not infinite and as such it is reasonable for the service to gracefully decline an unreasonable rate of requests from a single party in order to maintain a reasonable level of service for others.

- **Continuity**. It must be straightforward to apply updates to the service and these should not result in a noticeable break in responsiveness from the user's perspective.

- **Standard access**. External input and output to the translation modules occurs via a well defined API.

The effort required to create a platform from scratch, successfully addressing the above considerations would be enormous. Cloud computing platforms have developed around the need for institutions to create and maintain such platforms. Amazon Web Services (AWS) has emerged as a platform offering a diverse selection of cloud-based tools. Presented here is the architecture of the translation platform, implemented by the BBC on behalf of the GoURMET project and built entirely using AWS cloud-based solutions.

Whilst AWS is perhaps most known as a provider of Virtual Machines (VMs) via the EC2 product,[2] the platform presented here is not developed for and run on a VM. Development of a monolithic system for deployment on one or more VMs would still require considerable software development effort. Instead the platform described this this section combines a set of self-contained, individual AWS products such that each manages a separate facet of the platform. Software development work is therefore minimised and limited in the main to:

- the setup of each AWS component
- developing the behaviour for the component so it completes a specific task
- any development required to allow components to communicate

There are multiple cloud service providers available that could have been used to implement this Architecture including Google Cloud Platform[3] and Microsoft Azure[4]. Neither the project nor systems integrators endorse any specific cloud platform. The developers performing the systems integration were most familiar with AWS. However, it was still important to assess the feasibility of other platforms to ensure there is not a compelling reason to switch to an alternative, although none were found at that time.

The remainder of this section describes the architecture and details of the platform. The service provided by the platform is accessed via an API, the details of which are found in the following Section 4.

3.1 Detailed System Architecture

This platform is hosted on AWS and uses pre-existing services and components to create an architecture which is secure, robust and able to scale.

Key to building on AWS in this manner is the use of the AWS Cloudformation[5] service, which allows infrastructure to be defined using Cloudformation templates. A Cloudformation template is an example of infrastructure as code. By defining a template, a record of infrastructure is created that can be version controlled, provides visibility of the architecture and allows the system to be easily recreated from scratch.

A summary of the components and flow to fulfil a translation request is shown in Figure 3 and explained as follows.

A translation request will be initially handled by AWS API Gateway (Section 3.1.1), which is the user facing part of the architecture. The request is then passed to an AWS Lambda (Section 3.1.2,

[2] https://aws.amazon.com/ec2/
[3] https://cloud.google.com/
[4] https://azure.microsoft.com
[5] https://aws.amazon.com/cloudformation/

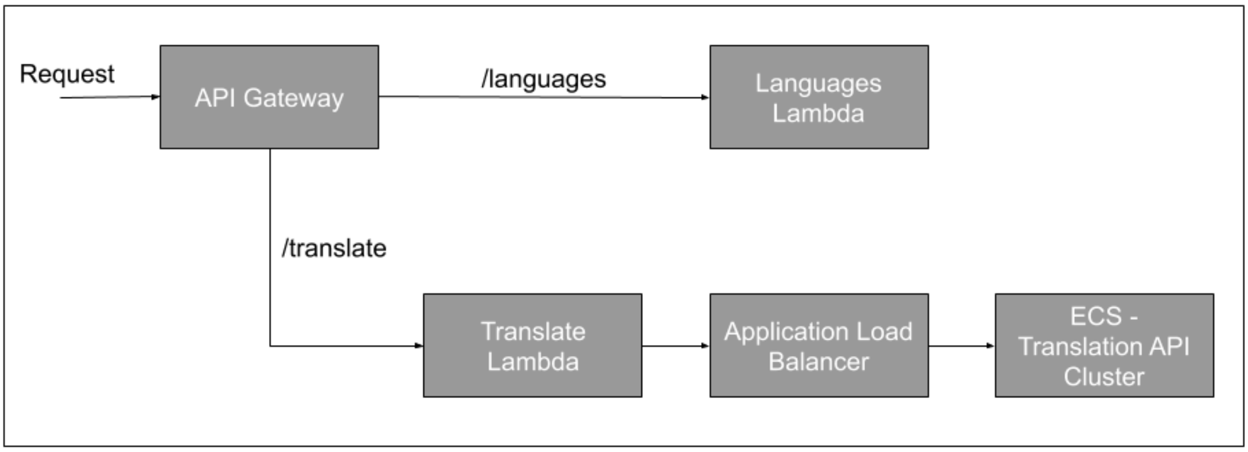

Figure 3: High level architecture of translation service platform

which acts as a bridge to an AWS Load Balancer (Section 3.1.3), which will route traffic to the correct translation model running in AWS Elastic Container Service (ECS) (Section 3.1.4). The model in ECS will perform the translation and the response travels back up the stack to be served to the requesting client by API Gateway. A more in depth explanation of the roles of the specific components is outlined in the following subsections.

3.1.1 AWS API Gateway

API Gateway[6] is an AWS managed service for creating APIs. The service is used to manage exposure of the MT models to the public internet as outlined in the previous section. In this case, the API is a REST API where the interface is defined using the OpenAPI specification standard[7]. As API Gateway is a managed service, it is easy to dynamically scale depending on traffic, and this service already implements multiple features for security, resilience and API life-cycle. This makes development of the user facing interface far quicker than starting from scratch.

As AWS API Gateway is a managed service one of its primary purposes is to reduce the challenges around setting up and maintaining an API, therefore the service makes certain assumptions about a user's requirements. As a result there are a number of standard API features that are either enabled by default or are simple to enable that do not require the API maintainer to understand the underlying implementation in order to use them. This includes:

- HTTPS by default
- Deployment of specific API stages. This allows the release and maintenance of multiple versions of the API in parallel and is useful if it is necessary to release breaking changes as it enables a more elegant migration of users and clients from the old service to the new.
- Standard handling of Error responses (e.g. invalid API Key, bad request body etc.) with the option configure the HTTP Response and set a custom status code, headers and/or body.
- Usage plans to set request rate, burst rate and monthly request quota

[6] https://aws.amazon.com/api-gateway/
[7] https://swagger.io/resources/open-api/

- Custom URL
- Authentication and authorisation of users
- API Key management
- Logs and metrics on API usage

API Keys API Keys are used by this platform for two reasons:

1. To restrict access to the API to authorised users only
2. To restrict the number and rate of requests an authorised user can make to ensure a single party cannot overload the platform with requests

API Gateway also offers a service for creating and managing Usage Plans and by extension API Keys. Through the Usage Plan request rate, burst rate and a monthly request quota can be specified. To enforce this Usage Plan the use of API Keys must be enabled. API Keys are created and managed through AWS. This includes the generation of a specific API key, associating the key with a specific usage plan and revoking the key if necessary.

3.1.2 AWS Lambda

AWS Lambda[8] offers serverless technology, which removes the need to maintain a server in order to run code. This makes it ideal for running short-lived tasks. A Lambda is created only when there is a need to execute the code and destroyed when that need no longer exists. This is good for both cost and dynamic scaling of services.

In the translation platform, the role of the Lambda is to route traffic from API Gateway to the Load Balancer.[9]. The Lambda is also responsible for providing enough information to the Load Balancer to route the traffic to the correct translation model. Introducing a Lambda also allows the Load Balancer to live within a private network and not be exposed to the public internet. This means that traffic to the Load Balancer and, by extension, the MT models is controlled and managed via API Gateway.

3.1.3 AWS Load Balancer

The Load Balancer functions at the application layer of the OSI (Open Systems Interconnection) model. The Load Balancer is listening for traffic on a collection of specific ports, where each port corresponds to a different translation model, and will route traffic from each port to a Target Group where there is one Target Group per translation model. The Target Group is made up of services that can receive the traffic. In this case, these are AWS ECS Tasks within an ECS Service. The Load Balancer will balance incoming traffic across all tasks within the Target Group as shown in Figure 4.

[8] https://aws.amazon.com/lambda/
[9] https://aws.amazon.com/elasticloadbalancing/

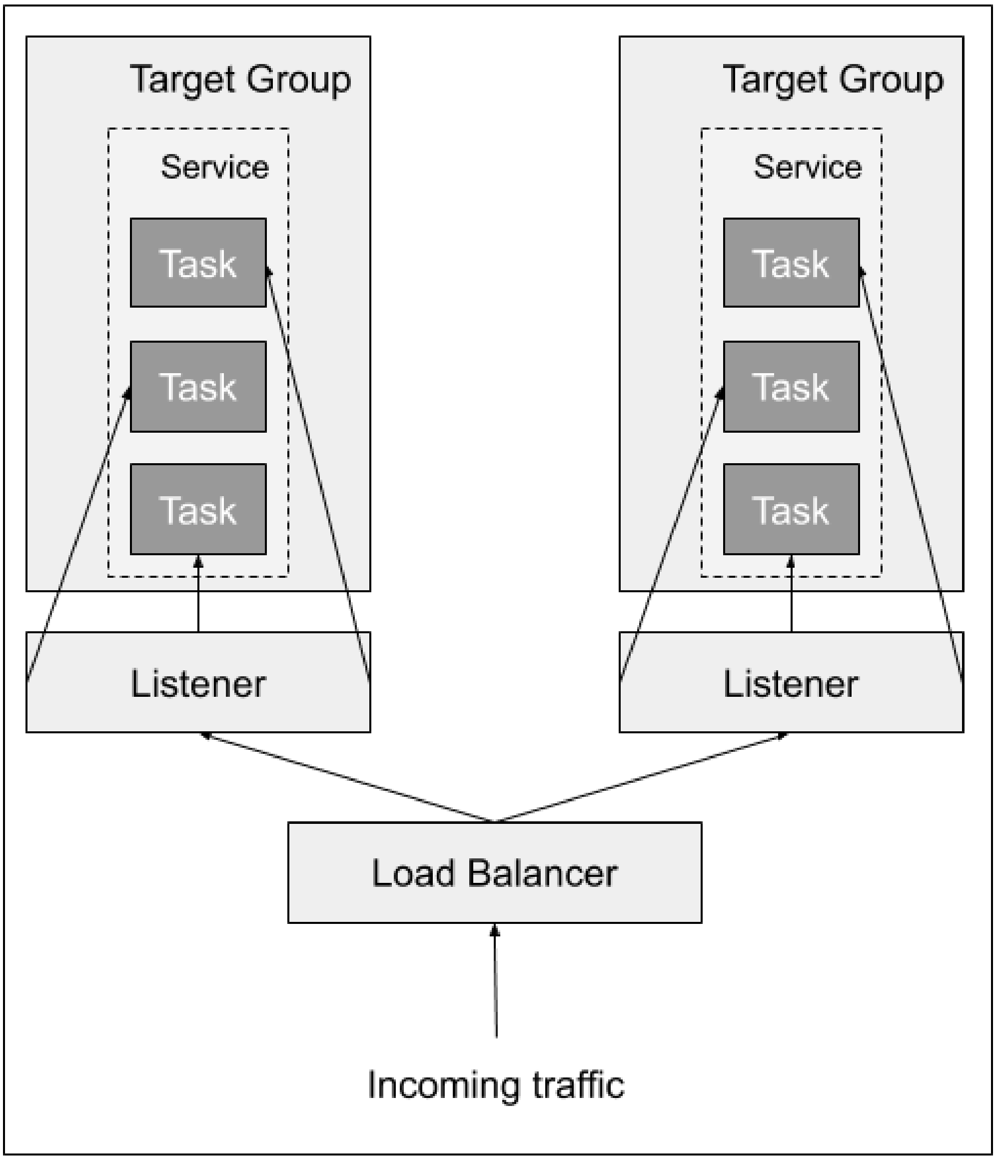

Figure 4: Routing traffic with a Load Balancer

3.1.4 AWS Elastic Container Service

The role of the ECS[10] is to run containers that contain the MT models.

All MT models are delivered as Docker images, which are definitions of how to create a container. This definition includes, but is not limited to, the operating system, programs installed, ports exposed, environment variables and file system. Containers provide an isolated environment in which the application may run, as defined in the Docker image. See Section 2 for details regarding the specifics of building a compliant container for this translation service.

Multiple containers can run on a single physical machine to allow an efficient sharing of resources. ECS is a service to manage how containers run as well as the infrastructure they will run on.

[10] https://aws.amazon.com/ecs/

The specific architecture of ECS is shown in Figure 5. An AWS Cluster has been created which defines the hardware the containers will run on. There are two options available for the AWS cluster: EC2 or Fargate. The EC2 approach requires the set up and maintenance of the hardware that ECS will run on. This includes choosing a Virtual Machine with sufficient memory and compute power to handle the number of running containers in the ECS Cluster as well as the general maintenance that comes with running a machine (installing appropriate software, managing patches and security updates etc.). Fargate is a managed service for providing hardware for the ECS Cluster. When using Fargate it is not necessary the maintain a Virtual Machine as AWS is responsible for maintaining and providing the compute power and memory on demand whenever a container is created within the ECS Cluster. This removes the overhead of maintaining a VM and calculating upfront the requirements of that machine in order to handle the number of running containers in the Cluster. The Fargate on demand approach makes scaling the system far simpler and cost effective as it makes it possible to only pay for compute and memory used. Due to the small size of the team and the desire to be able to scale the system easily depending on demand Fargate was chosen as the underlying infrastructure for the AWS Cluster.

An AWS Task Definition has been created for each MT Docker image. The Task Definition defines the properties of a container. This includes but is not limited to which Docker image to use and where to pull the image from, how much CPU power and memory to allocate the container and any AWS IAM Roles[11] the container needs. Containers created using the Task Definitions are referred to as Tasks in AWS. The Tasks have been created within a Service. The Service maintains a specified number of instances of a Task and allows for the number of Tasks to be scaled up or down according to load on the system. The Service also health-checks Tasks and destroys and replaces unhealthy ones. ECS provides logs and metrics about the Cluster as well as the Tasks and Services contained within it this allows the monitoring of the health of the system including number of tasks running, CPU Utilised and Memory Utilised.

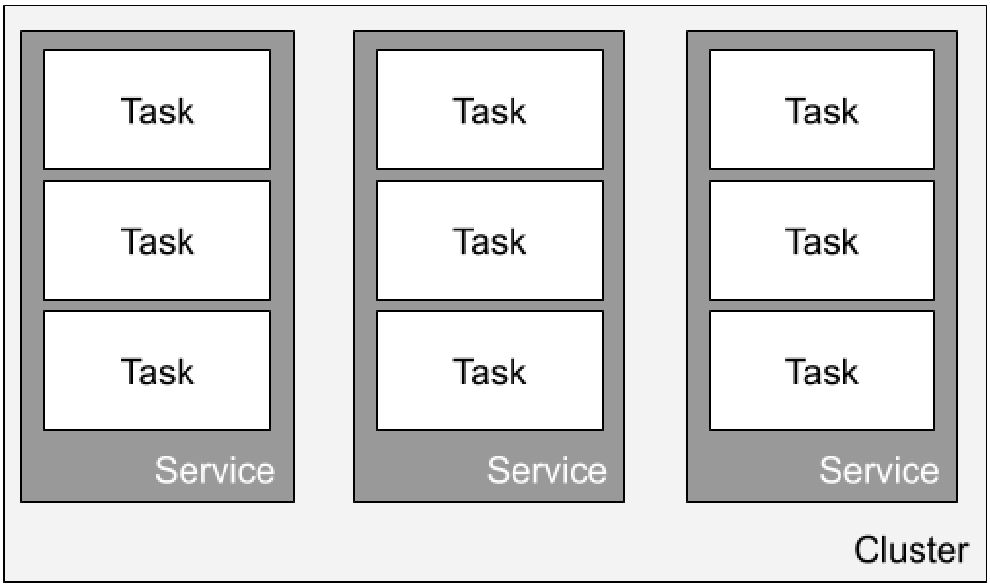

Figure 5: Architecture of an ECS Cluster

[11] https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles.html

3.2 Security, Access Management and Request Rate Limiting

Security is important for any service available on the public internet, as the service is vulnerable to attacks from malicious users.

An evaluation of the platform architecture design concluded that:

- No sensitive information can be exposed if an API key was to become compromised.
- With access to the running system, a malicous actor could intercept translation input or output which the client might rightly assume to be secret. The malicious actor could alter the output of the translation service back to the client. The impact of a security breach such as this could be high, but the risk of a malicous 3rd party gaining access to the infrastructure itself is low.
- Interception of unencrypted traffic to and from the service is a moderate risk. Content that news organisations submit for translation may be sensitive in nature. This may include personal data, the content of ongoing journalistic investigations or as yet unpublished news content which may be considered if high value to the organisation. News organisations must have confidence that this traffic is appropriately secured and as such encrypted communication must be established between client and trabslation service.
- A DDoS (Distributed Denial of Service) attack represents a large risk. A DDoS attack overwhelms a system with requests to stop it being able to handle legitimate requests. In this case, the scalable nature of the architecture would make it possible to require the services to use a large number of additional resources to handle malicious traffic increases, if the service is not properly secured. For the translation platform described herein, the impact to clients due to increased response times may be moderate, but the financial costs that can be caused by an unbounded automatic increase in resource use could be severe.

To mitigate the above risks, the service is secured as follows:

- Using HTTPS: All traffic is served over HTTPS by default with API Gateway managing the certificate.
- Using API Keys
- Using Usage Plans: Usage plans are tied to specific API keys and add a throttling limit and quota limit.
- Sanitising Input: Ensure required request inputs are included and that the body matches the JSON schema and request model.
- Limiting payload size: API Gateway has an upper payload limit of 10 MB[12]. This allows more than enough capacity to handle translation of news articles whilst setting an upper limit to prevent malicious attacks where the system is overloaded by requests with large payloads.

[12] https://docs.aws.amazon.com/apigateway/latest/developerguide/limits.html

- Security of the infrastructure itself (i.e. access to the lambdas, running Docker images, etc) is secured at a number of levels. Explicit permission must be granted for a user to access the AWS account containing the platforms's infrastructure. All users must identify themselves by means of a certificate.

AWS's API Gateway service was one reason to favour AWS when building this platform. API Gateway natively provides many of the security features listed above. This means that the in-depth knowledge, development and testing needed to implement security features is not required, as they do not need to be implemented from scratch. Furthermore, API Gateway is widely used in industry and is actively maintained which means that security flaws are detected early and patched quickly.

3.3 Scalability

It is important to have a service that can scale dynamically in response to changes in volume of traffic. There are two points of scalability in architecture shown in Figure 3.

The first is using ECS. A Service determines the number of instances created from a specific Task Definition that are running at any one time, and this allows the translation platform to scale up to accommodate more traffic. The Load Balancer will distribute the traffic amongst the growing number of Tasks available to fulfil the requests.

The other point of scalability is the Lambda. Serverless technology is designed to be flexible, as it is not the responsibility of the Lambda creator to define the hardware it will run on or to ensure sufficient compute resources are available for it to run. As a result, Lambdas can be automatically initiated as needed to handle increases in traffic without the need to predict traffic spikes and provision hardware to handle these. The API Gateway is designed to handle large amounts of traffic hitting the translation platform and using Lambdas to fulfil these requests allows the system to handle these traffic increases.

The final consideration regarding scalability is the ability to automate this scaling. AWS provides Cloudwatch Alarms to monitor and automatically respond to changes in the system under monitoring. This allows the translation platform to alert in response to changes in traffic and use of resources. These alerts can be used to handle these changes without manual intervention.

3.4 Deploying MT Models

Section 2 described in depth how translation modules are supplied to the integrators.

With regards to the integration and running of these in the translation platform architecture, the Docker images, described in Section 2 are hosted on AWS using the AWS Elastic Container Registry (ECR).[13] Docker images can be hosted as repositories on any service that provides a Docker registry for example Docker Hub[14]. Registries provide a central place to store images and the ability to only allow authorised users access to those images. Repositories use tags to allow the images to be version controlled. When a Task is created in ECS, the image is pulled from ECR then creates a Docker container using this image.

[13] https://aws.amazon.com/ecr/
[14] https://hub.docker.com/

4 Translation API

The translation platform as described in the preceding section makes available translations as a service. Access to this service is via a RESTful API. The purpose of this API is to provide a consistent, reliable and functional interface for interaction with the translation models. The API described in this section is designed to be as simple as possible. This promotes ease of integration for clients and simplifies maintenance and updates for the systems integrators. The RESTful paradigm used also promotes straightforward integration via standard and well tested libraries. In this model, access to the translation service is via standard HTTP methods, with translations returned as HTTP responses. Errors are notified to the client via standard HTTP error codes which are profiled by the specification to have specific meanings in the context of the request.

The remainder of this section provides a summary of the most important aspects of the API. For the full detailed specification an internal specification document ”Specification 5.2 – Translation API specification”Coleman and Secker (2020) is available as a companion document.

4.1 Standards for Translation APIs

Ideally the translation service described here would implement a standard translation API format for access. No internationally published standards (open or otherwise) appear to exist for the input and output to/from such a translation service. A recognised industry standard would be ratified by a number of interested parties, typically forming a working group with members drawn from across industry, and published through a recognised body (such as ISO, IEEE, ETSI, MPEG, and so on), with the view of cross-industry adoption.

A number of independently published specifications exist, mainly published by commercial translation service providers. Notable examples include Amazon https://docs.aws.amazon.com/translate/latest/dg/translate-dg.pdf and Google via the Cloud Platform product https://cloud.google.com/translate/docs/reference/rest/v2/translate. Individual specifications such as these are not standards as defined above. Rather they are proprietary and subject to change without consultation. They cannot be used as a specification for this service to implement. However, they may be used as inspiration and have been be used as examples of good practice when defining the specification contained in this section.

4.2 Security

All calls to the API detailed in this section take place over HTTPS.

In order to use any API endpoint the request must include a valid API Key. Any request without an API key or an invalid API key will be rejected.

Any data provided by the user, for example payloads on POST request or query parameters, should be sanitised. The sanitisation process is required to stop malicious inputs from the user. If necessary illegal characters will be removed or the input rejected. If this is the case a suitable error code and message will be returned to the client. Requests containing request parameters that cannot be sanitised will be rejected in the same way.

The size of the payload provided in the body of a request should also be checked to ensure that the JSON will not overload the parser handling inputs. Requests with payloads larger than 10MB will be rejected by the service.

4.2.1 API Key Management

All requests must include a valid API key, supplied as defined in Section 4.3. API Keys will be issued and managed by the systems integrators (at the time of writing, this is the BBC), with the expectation that key holders will take reasonable precautions to keep this key secure i.e. not checking the key into a public repository or sharing it with others. Keys may be issued to an individual client or to an institution, whichever is appropriate.

Keys which are compromised, have been misused or are otherwise no longer needed may be revoked.

Key generation and management is handled using API Gateway (Section 3.1.1), which has the advantage of integrating these key management tasks with the API access mediation itself.

4.2.2 Rate Limiting, Throttling & Consumption Quotas

The purpose of rate limiting and throttling is to ensure the number of requests that a client, using a single API key, cannot overwhelm the service with requests to the detriment of of other clients. With a service such as this which scales based on demand, it also places an upper bound on the (theoretical) maximum capacity required to service all clients. In the case of a DDoS attack using a compromised key (See Section 3.2) imposing such a limit can be a prudent measure for mitigating impact.

Each key can be assigned its own individual quota. There are three metrics that are set on API Keys, request rate, burst rate and number of requests that can be made per month. Quotas can be adjusted as required after the key is issued.

In the case of a quota being exceeded with a specific key, the client will receive a `429 Too Many Requests` HTTP status code as a response to a request. The client should wait before making the request again or in the case that they have exceeded their allowed requests for the month wait for the start of the next month. It is recommended that clients proactively manage their request rate according to the rate limiting parameters set for their individual key.

4.3 API details

Methods supported by the API are summarised below. The full specification, required for integration with the service, is found in the companion document Coleman and Secker (2020).

{`base`}, is used as a placeholder for the location of the translation service.

All calls to the API use HTTPS and are thus made on the standard port, TCP 443.

4.3.1 Interpretation of HTTP Response Codes

All calls to the API will return an HTTP status code. A successful request will be indicated by a HTTP response code 200.

4XX responses indicate client-side errors. After taking the necessary action to correct the error, the client can make the request again. 5XX errors are server-side. It is safe for the client to retry the request, but there is no guarantee the service will have recovered. When sending a 4XX or

5XX error code, the service will also return a JSON response in the response body describing the details of the error if available.

A summary of how response codes should be interpreted in the context of this service is listed below.

`400 Bad Request` - the input from the client is invalid. For example missing parameters, invalid language code, incorrectly formed body, unsupported media type etc.

`401 Unauthorized` - the client cannot be authenticated. This may be triggered by the API key being invalid or missing.

`405 Method Not Allowed` - the client has attempted to perform a request using an unsupported HTTP method.

`413 Payload Too Large` - the request is larger than the system is willing or able to process.

`429 Too Many Requests` - the client has sent too many requests in a given time period and is subject to throttling.

`500 Internal Server Error` - there is an unexpected error on the server side. This includes but is not limited to invalid server side configuration or integration, failure of the authoriser system or timeout of a server side system,

`502 Bad Gateway` - there is an invalid response from an upstream server.

`503 Service Unavailable` - the server is unavailable.

`504 Gateway Timeout` - an upstream server does not send back a timely response.

4.4 Versioning

Wherever possible changes made to the API will be backwards compatible. Provision has been made for managing breaking changes via versioning.

The standard request path contains a version identifier. i.e.

```
{base}/{version_identifier}/
```

If it becomes necessary to to introduce breaking changes to one version of the API, this will be managed by incrementing the version number. In this case, users will be notified that the previous version of the API is being retired. Two API versions will be maintained in parallel for a period of time to allow migration of services to the new version.

The increment of a version number will be avoided as much as possible as it is undesirable in terms of effort and cost to maintain two versions of the service.

4.4.1 Supported API Methods

Supported Languages

Description: Query the service for a list of supported language pairs.

Method location: {base}/{version_identifier}/languages

HTTP Method Type: GET

URL parameters: none

HTTP headers: `x-api-key` - a valid API Key

Details: A successful request will return a JSON response containing details of supported language pairs and directions. The JSON will have the form:

```
{
    "languagePairs": array[LanguagePair],
}
```

Where the LanguagePair is an Object:

```
{
    "source": string,
    "target": string
}
```

The returned strings are 2-character (ISO 639-1 alpha-2 code) language codes representing the language.

Translate

Description: Translate a string of text from one language to another.

Method location: {base}/{version_identifier}/translate

HTTP Method Type: POST

URL parameters: None

HTTP headers: `x-api-key` - a valid API Key

Request Body: The body of the request must be JSON and have the following shape where all parameters are present:

```
{
    "q": string,
    "source": string,
    "target: string,
}
```

Where:

q: text to translate

source: is the 2-character (ISO 639-1 alpha-2 code) language code representing the source language

target: is the 2-character (ISO 639-1 alpha-2 code) language code representing the target language

Response: A successful translation will include an HTTP response code 200 and the following JSON in the body:

```
{
    "translatedText": string,
    "source": string,
    "target: string,
}
```

5 Demonstrator User Interface

The API, as described previously allows translation between the language pairs created by the project. While an API is useful when integrating the translation technology into software it is not a very user-friendly way to demonstrate the work done by the GoURMET consortium members either internally to stakeholders, or externally at events.

In order to make the translation models available to a wide audience, a simple translation user interface was created which uses the translation models developed within the project. A web-page based user interface such as this can provide a simple way to test the translation models and demonstrate the research technology. The integration team worked with a UX Designer to build a user facing graphical interface. Building this interface gives all members of the consortium an easy way to interact with all the translation models available as well as demo their work to others. Being web-based there is no set up required before use.

The desktop version of the UI is shown in Figure 6.

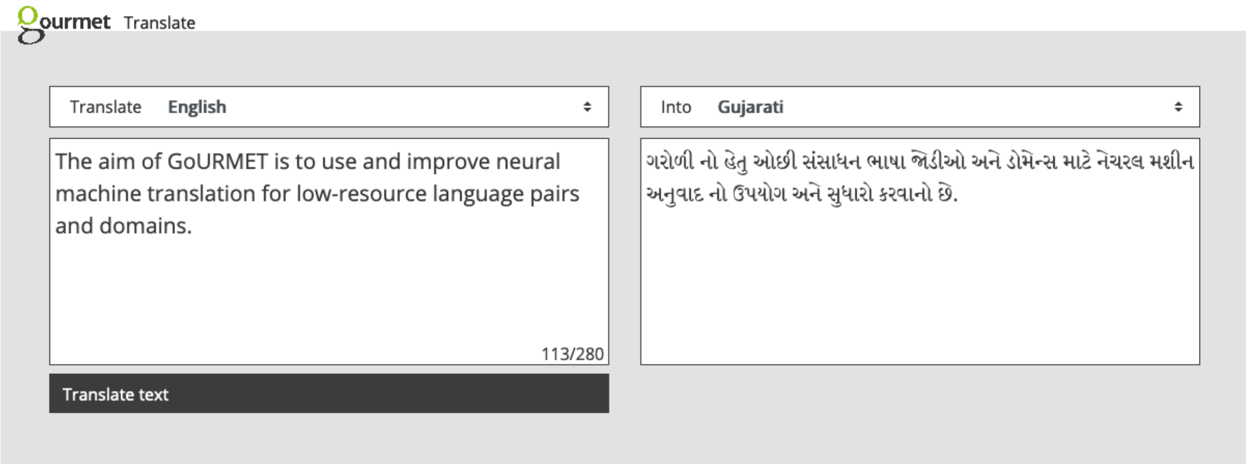

Figure 6: GoURMET Translation User Interface

One of the goals when building this UI was to make it possible to carry out demos of the translation technology anywhere, and as a result the team also ensured that the UI had a mobile friendly view, shown in Figure 7. The UI was built using the React Framework for the front end, and Node with the Express framework for the back end. Both the front end and back end are using TypeScript. The code itself runs on AWS Infrastructure, with simplified architecture shown diagrammatically in Figure 8

The application runs on an EC2 instance which sits within an autoscaling group. The autoscaling group is responsible for scaling up and down the number of instances as required to deal with the amount of traffic received by the service (i.e. the number of people using the UI simultaneously). The autoscaling group is also responsible for monitoring the EC2 instances. This involves checking at regular intervals that all instances pass a health check and replacing any EC2 instances that does not with a new virtual machine. The load balancer is responsible for the distribution of traffic across available EC2 instances.

All instances communicate with the translation service using the API as described previously in Section 4.

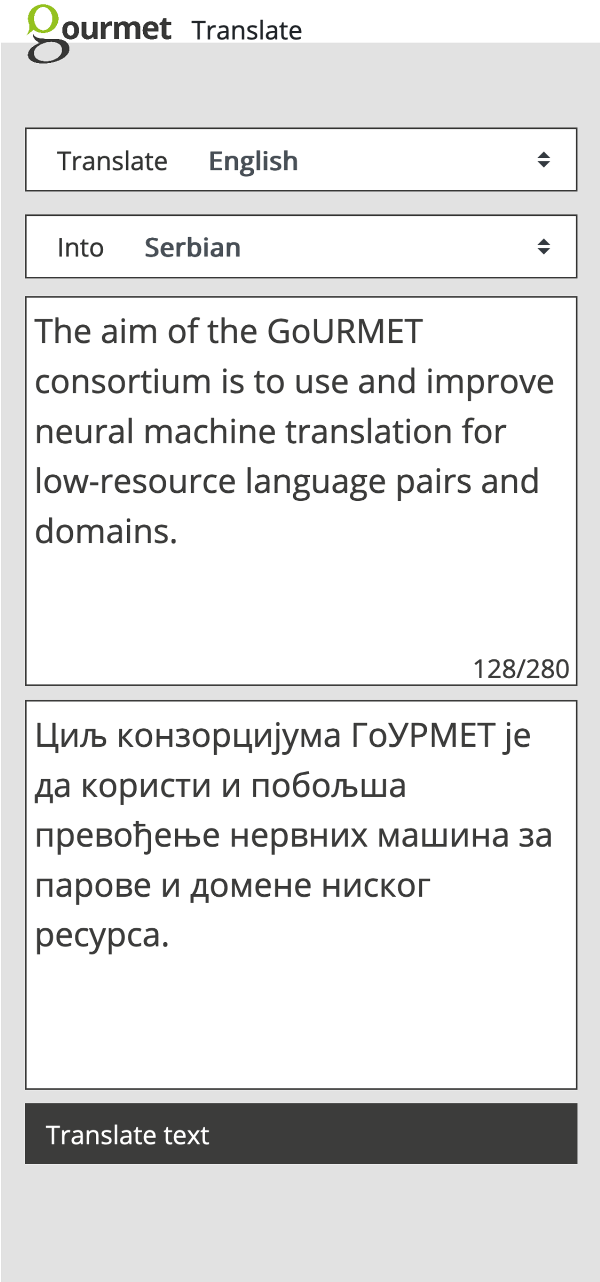

Figure 7: Translation UI on mobile

This UI has proved a valuable way to start conversations about the work the consortium is doing. It has also highlighted why building an API to access the translation models is so important - by providing a single platform it is easy to build multiple clients that use the underlying technology through a standard REST API rather than having to integrate the translation models directly into an application.

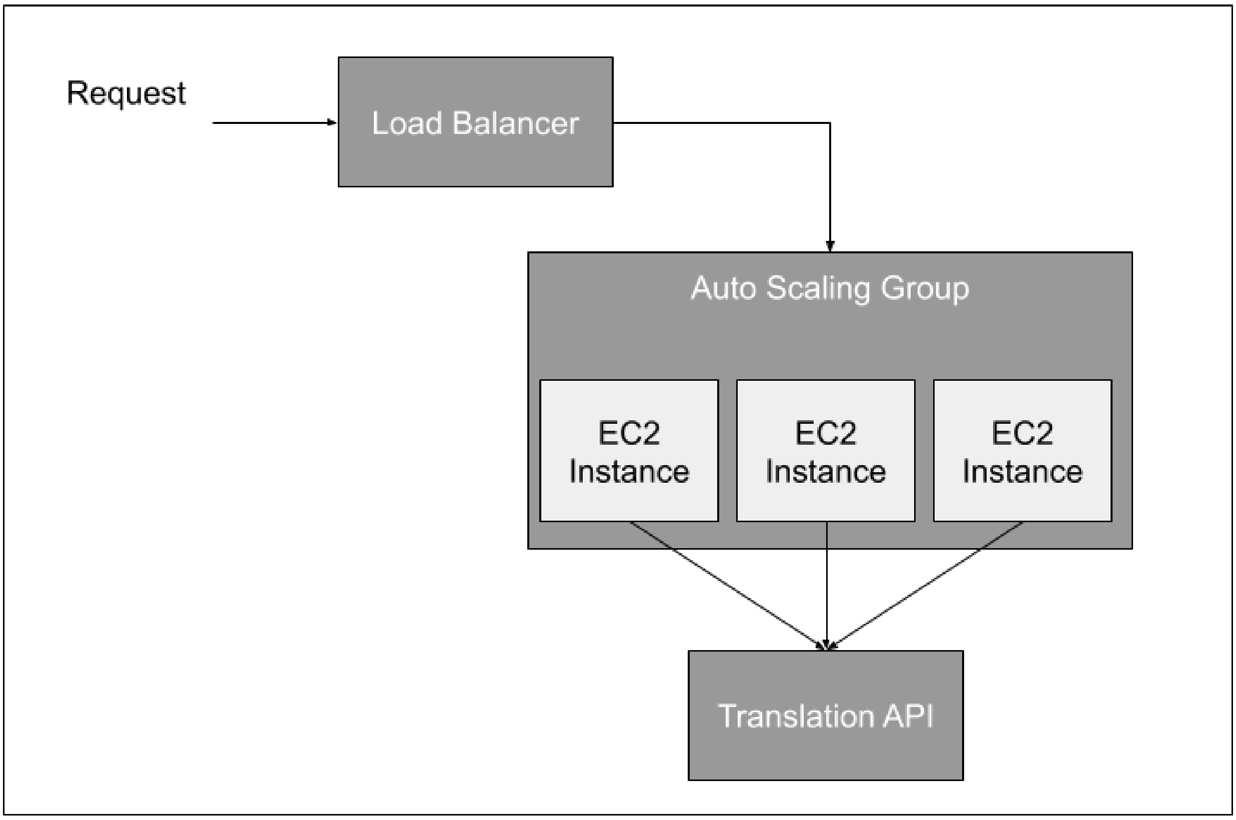

Figure 8: AWS Infrastructure for Translation UI

6 Research Outputs

6.1 Publications

These papers contain information regarding the Integration aspects of the GoURMET project.

- Susie Coleman, Andrew Secker, Rachel Bawden, Barry Haddow, and Alexandra Birch. Architecture of a scalable, secure and resilient translation platform for multilingual news media. In *Proceedings of the 1st International Workshop on Language Technology Platforms*, pages 16–21, Marseille, France, May 2020a. European Language Resources Association. ISBN 979-10-95546-64-1. URL https://www.aclweb.org/anthology/2020.iwltp-1.3

- Rachel Bawden, Nikolay Bogoychev, Ulrich Germann, Roman Grundkiewicz, Faheem Kirefu, Antonio Valerio Miceli Barone, and Alexandra Birch. The university of Edinburgh's submissions to the WMT19 news translation task. In *Proceedings of the Fourth Conference on Machine Translation (Volume 2: Shared Task Papers, Day 1)*, pages 103–115, Florence, Italy, August 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-5304. URL https://www.aclweb.org/anthology/W19-5304

- Felipe Sánchez-Martínez, Víctor M. Sánchez-Cartagena, Juan Antonio Pérez-Ortiz, Mikel L. Forcada, Miquel Esplà-Gomis, Andrew Secker, Susie Coleman, and Julie Wall. An English-Swahili parallel corpus and its use for neural machine translation in the news domain. In *Proceedings of the 22th Annual Conference of the European Association for Machine Translation*, pages 299–308, Online Conference, November 2020

6.2 Datasets

The training data sets are covered in Deliverable D1.3.

7 Conclusion

In order to support multiple tools which require translation technology, a platform providing translation as a service is the preferred solution. This document has described the elements that have been put in place by GoURMET WP5 in order to support this.

The details of the translation models supplied for integration were described. The requirements around the delivery of these to the integrators were outlined followed by the architecture of the translation service which makes those translation models available. An API onto the translation service was detailed and the an example user interface, suitable for demonstration purposes was reported.

The translation platform described herein will form the basis of tools and prototypes to be created by and tested within the BBC and DW. This will be done under Task T5.4 from the GoURMET WP5 Description of Work and reported in the future D5.5 deliverable. In the future, these prototypes will also support further and real-world evaluation of the underlying MT systems.

References

Ž. Agić and I. Vulić. JW300: A wide-coverage parallel corpus for low-resource languages. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3204–3210, Florence, Italy, July 2019. doi: 10.18653/v1/P19-1310. URL https://www.aclweb.org/anthology/P19-1310.

D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014.

Marta Bañón, Pinzhen Chen, Barry Haddow, Kenneth Heafield, Hieu Hoang, Miquel Esplà-Gomis, Mikel Forcada, Amir Kamran, Faheem Kirefu, Philipp Koehn, Sergio Ortiz-Rojas, Leopoldo Pla, Gema Ramírez-Sánchez, Elsa Sarrías, Marek Strelec, Brian Thompson, William Waites, Dion Wiggins, and Jaume Zaragoza. Paracrawl: Web-scale acquisition of parallel corpora. In *Proceedings of Annual Meeting of the Association for Computational Linguistics*, 2020.

Loïc Barrault, Ondřej Bojar, Marta R. Costa-jussà, Christian Federmann, Mark Fishel, Yvette Graham, Barry Haddow, Matthias Huck, Philipp Koehn, Shervin Malmasi, Christof Monz, Mathias Müller, Santanu Pal, Matt Post, and Marcos Zampieri. Findings of the 2019 conference on machine translation (WMT19). In *Proceedings of the Fourth Conference on Machine Translation (Volume 2: Shared Task Papers, Day 1)*, pages 1–61, Florence, Italy, August 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-5301. URL https://www.aclweb.org/anthology/W19-5301.

Rachel Bawden, Nikolay Bogoychev, Ulrich Germann, Roman Grundkiewicz, Faheem Kirefu, Antonio Valerio Miceli Barone, and Alexandra Birch. The university of Edinburgh's submissions to the WMT19 news translation task. In *Proceedings of the Fourth Conference on Machine Translation (Volume 2: Shared Task Papers, Day 1)*, pages 103–115, Florence, Italy, August 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-5304. URL https://www.aclweb.org/anthology/W19-5304.

O. Bojar, C. Federmann, M. Fishel, Y. Graham, B. Haddow, P. Koehn, and C. Monz. Findings of the 2018 conference on machine translation (WMT18). In *Proceedings of the Third Conference on Machine Translation: Shared Task Papers*, pages 272–303, Belgium, Brussels, October 2018a. Association for Computational Linguistics. doi: 10.18653/v1/W18-6401. URL https://www.aclweb.org/anthology/W18-6401.

Ondrej Bojar, Christian Federmann, Mark Fishel, Yvette Graham, Barry Haddow, Matthias Huck, Philipp Koehn, and Christof Monz. Findings of the 2018 conference on machine translation (wmt18). In *Proceedings of the Third Conference on Machine Translation, Volume 2: Shared Task Papers*, pages 272–307, Belgium, Brussels, October 2018b. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/W18-6401.

David Chiang. A hierarchical phrase-based model for statistical machine translation. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 263–270. Association for Computational Linguistics, 2005.

Susie Coleman and Andrew Secker. GoURMET: Specification 5.2 – translation api specification, 2020.

Susie Coleman, Andrew Secker, Rachel Bawden, Barry Haddow, and Alexandra Birch. Architecture of a scalable, secure and resilient translation platform for multilingual news media. In *Proceedings of the 1st International Workshop on Language Technology Platforms*, pages 16–21, Marseille, France, May 2020a. European Language Resources Association. ISBN 979-10-95546-64-1. URL https://www.aclweb.org/anthology/2020.iwltp-1.3.

Susie Coleman, Andrew Secker, Andrew Blaney, and Manish Lad. GoURMET: Specification 5.1 - translation module and version control specification, 2020b.

G. De Pauw, P.W. Wagacha, and G.-M. De Schryver. Exploring the SAWA corpus: collection and deployment of a parallel corpus English–Swahili. *Language resources and evaluation*, 45(3): 331, 2011.

Chris Dyer, Victor Chahuneau, and Noah A. Smith. A simple, fast, and effective reparameterization of IBM model 2. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 644–648, Atlanta, Georgia, June 2013. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/N13-1073.

Bryan Eikema and Wilker Aziz. Auto-encoding variational neural machine translation. In *Proceedings of the 4th Workshop on Representation Learning for NLP (RepL4NLP-2019)*, pages 124–141, Florence, Italy, August 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-4315. URL https://www.aclweb.org/anthology/W19-4315.

Miquel Espla-Gomis and Mikel Forcada. Combining content-based and url-based heuristics to harvest aligned bitexts from multilingual sites with bitextor. *The Prague Bulletin of Mathematical Linguistics*, 93(1):77–86, 2010.

Y. Graham, T. Baldwin, A. Moffat, and J. Zobel. Can machine translation systems be evaluated by the crowd alone. *Natural Language Engineering*, 23(1):3–30, 2016.

Kenneth Heafield. KenLM: Faster and smaller language model queries. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, pages 187–197, Edinburgh, Scotland, July 2011. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/W11-2123.

Kenneth Heafield, Ivan Pouzyrevsky, Jonathan H. Clark, and Philipp Koehn. Scalable modified Kneser-Ney language model estimation. In *ACL (2)*, pages 690–696, 2013.

V.C.D. Hoang, P. Koehn, G. Haffari, and T. Cohn. Iterative back-translation for neural machine translation. In *Proceedings of the 2nd Workshop on Neural Machine Translation and Generation*, pages 18–24, Melbourne, Australia, July 2018. URL https://www.aclweb.org/anthology/W18-2703.

Matthias Huck and Hermann Ney. Pivot lightly-supervised training for statistical machine translation. In *Proc. 10th Conf. of the Association for Machine Translation in the Americas*, pages 50–57, 2012.

Melvin Johnson, Mike Schuster, Quoc V. Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda Viégas, Martin Wattenberg, Greg Corrado, Macduff Hughes, and Jeffrey

Dean. Google's multilingual neural machine translation system: Enabling zero-shot translation. *Transactions of the Association for Computational Linguistics*, 5:339–351, 2017. doi: 10.1162/tacl_a_00065. URL https://doi.org/10.1162/tacl_a_00065.

Marcin Junczys-Dowmunt. Dual conditional cross-entropy filtering of noisy parallel corpora. In *Proceedings of the Third Conference on Machine Translation: Shared Task Papers*, pages 888–895, Belgium, Brussels, 2018. URL https://www.aclweb.org/anthology/W18-6478.

Marcin Junczys-Dowmunt, Roman Grundkiewicz, Tomasz Dwojak, Hieu Hoang, Kenneth Heafield, Tom Neckermann, Frank Seide, Ulrich Germann, Alham Fikri Aji, Nikolay Bogoychev, André F. T. Martins, and Alexandra Birch. Marian: Fast Neural Machine Translation in C++. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, ACL'18, pages 116–121, Melbourne, Australia, 2018.

Young Jin Kim, Marcin Junczys-Dowmunt, Hany Hassan, Alham Fikri Aji, Kenneth Heafield, Roman Grundkiewicz, and Nikolay Bogoychev. From research to production and back: Ludicrously fast neural machine translation. In *Proceedings of the 3rd Workshop on Neural Generation and Translation*, pages 280–288, Hong Kong, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-5632. URL https://www.aclweb.org/anthology/D19-5632.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondrej Bojar, Alexandra Constantin, and Evan Herbst. Moses: Open Source Toolkit for Statistical Machine Translation. In *Proceedings of the Association for Computational Linguistics Companion Demo and Poster Sessions*, pages 177–180, Prague, Czech Republic, 2007a. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P/P07/P07-2045.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bojar, Alexandra Constantin, and Evan Herbst. Moses: Open Source Toolkit for Statistical Machine Translation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics*, ACL'07, pages 177–180, Prague, Czech Republic, 2007b.

Taku Kudo and John Richardson. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, Brussels, Belgium, November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-2012. URL https://www.aclweb.org/anthology/D18-2012.

Anoop Kunchukuttan. The IndicNLP Library. https://github.com/anoopkunchukuttan/indic_nlp_library/blob/master/docs/indicnlp.pdf, 2020.

Guillaume Lample and Alexis Conneau. Cross-lingual Language Model Pretraining. In *arXiv:1901.07291*, 2019. URL http://arxiv.org/abs/1901.07291.

R.V. Lim, K. Heafield, H. Hoang, M. Briers, and A.D. Malony. Exploring hyper-parameter optimization for neural machine translation on GPU architectures. *CoRR*, abs/1805.02094, 2018. URL http://arxiv.org/abs/1805.02094.

Yinhan Liu, Jiatao Gu, Naman Goyal, Xian Li, Sergey Edunov, Marjan Ghazvininejad, Mike Lewis, and Luke Zettlemoyer. Multilingual denoising pre-training for neural machine translation, 2020.

M. Nadejde, S. Reddy, R. Sennrich, T. Dwojak, M. Junczys-Dowmunt, P. Koehn, and A. Birch. Predicting target language CCG supertags improves neural machine translation. In *Proceedings of the Second Conference on Machine Translation, Volume 1: Research Papers*, pages 68–79, Copenhagen, Denmark, September 2017.

Franz Josef Och and Hermann Ney. A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29(1):19–51, 2003. doi: 10.1162/089120103321337421. URL https://www.aclweb.org/anthology/J03-1002.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of NAACL-HLT 2019: Demonstrations*, 2019.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: A Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, ACL'02, pages 311–318, Philadelphia, Pennsylvania, USA, 2002.

Martin Popel and Ondřej Bojar. Training tips for the transformer model. *The Prague Bulletin of Mathematical Linguistics*, 110(1):43–70, 2018.

Maja Popović. chrf++: words helping character n-grams. In *Proceedings of the second conference on machine translation*, pages 612–618, 2017.

Matt Post. A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191, Brussels, Belgium, October 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-6319. URL https://www.aclweb.org/anthology/W18-6319.

O. Press and L. Wolf. Using the output embedding to improve language models. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 157–163, Valencia, Spain, April 2017.

P. Qi, T. Dozat, Y. Zhang, and C.D. Manning. Universal dependency parsing from scratch. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 160–170, Brussels, Belgium, October 2018. URL https://nlp.stanford.edu/pubs/qi2018universal.pdf.

Víctor M. Sánchez-Cartagena, Marta Bañón, Sergio Ortiz-Rojas, and Gema Ramírez. Prompsit's submission to WMT 2018 parallel corpus filtering shared task. In *Proceedings of the Third Conference on Machine Translation: Shared Task Papers*, pages 955–962, Belgium, Brussels, October 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-6488. URL https://www.aclweb.org/anthology/W18-6488.

Víctor M. Sánchez-Cartagena, Juan Antonio Pérez-Ortiz, and Felipe Sánchez-Martínez. The Universitat d'alacant submissions to the English-to-Kazakh news translation task at WMT 2019. In *Proceedings of the Fourth Conference on Machine Translation (Volume 2: Shared Task Papers,* *Day 1)*, pages 356–363, Florence, Italy, August 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-5339. URL https://www.aclweb.org/anthology/W19-5339.

H. Schmid. Probabilistic part-of-speech tagging using decision trees. In *New methods in language processing*, page 154, 2013.

Holger Schwenk, Vishrav Chaudhary, Shuo Sun, Hongyu Gong, and Francisco Guzmán. Wikimatrix: Mining 135m parallel sentences in 1620 language pairs from wikipedia. *CoRR*, abs/1907.05791, 2019. URL http://arxiv.org/abs/1907.05791.

Andrew Secker, Julie Wall, Peggy van der Kreeft, and Susie Coleman. GoURMET: Deliverable 5.2 - use cases and requirements, 2019.

Andrew Secker, Julie Wall, Mikel L. Forcada, Barry Haddow, Antonio Miceli Barone, Susie Coleman, and Anna Blaziak. GoURMET: Deliverable 5.4 - initial progress report on evaluation, 2020.

R. Sennrich, B. Haddow, and A. Birch. Edinburgh Neural Machine Translation Systems for WMT 16. In *Proceedings of the First Conference on Machine Translation*, pages 371–376, Berlin, Germany, August 2016a. URL http://www.aclweb.org/anthology/W/W16/W16-2323.

R. Sennrich, A. Birch, A. Currey, U. Germann, B. Haddow, K. Heafield, A.V. Miceli Barone, and P. Williams. The University of Edinburgh's Neural MT Systems for WMT17. In *Proceedings of the Second Conference on Machine Translation, Volume 2: Shared Task Papers*, pages 389–399, Copenhagen, Denmark, September 2017. URL http://www.aclweb.org/anthology/W17-4739.

Rico Sennrich and Biao Zhang. Revisiting low-resource neural machine translation: A case study. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 211–221, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1021. URL https://www.aclweb.org/anthology/P19-1021.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Improving neural machine translation models with monolingual data. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 86–96, Berlin, Germany, August 2016b. Association for Computational Linguistics. doi: 10.18653/v1/P16-1009. URL https://www.aclweb.org/anthology/P16-1009.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural Machine Translation of Rare Words with Subword Units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, ACL'16, pages 1715–1725, Berlin, Germany, 2016c.

N. Silveira, T. Dozat, M.-C. de Marneffe, S. Bowman, M. Connor, J. Bauer, and C.D. Manning. A gold standard dependency corpus for English. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC-2014)*, pages 2897–2904, Reykjavik, Iceland, May 2014.

Felipe Sánchez-Martínez, Víctor M. Sánchez-Cartagena, Juan Antonio Pérez-Ortiz, Mikel L. Forcada, Miquel Esplà-Gomis, Andrew Secker, Susie Coleman, and Julie Wall. An English-Swahili parallel corpus and its use for neural machine translation in the news domain. In *Proceedings of the 22th Annual Conference of the European Association for Machine Translation*, pages 299–308, Online Conference, November 2020.

Jörg Tiedemann. Parallel data, tools and interfaces in opus. In *LREC*, 2012.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc., 2017.

Yiren Wang, Yingce Xia, Tianyu He, Fei Tian, Tao Qin, ChengXiang Zhai, and Tie-Yan Liu. Multi-agent dual learning. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HyGhN2A5tm.

Michał Ziemski, Marcin Junczys-Dowmunt, and Bruno Pouliquen. The united nations parallel corpus v1.0. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 3530–3534, Portorož, Slovenia, May 2016. European Language Resources Association (ELRA). URL https://www.aclweb.org/anthology/L16-1561.

Appendix A Translation Model Details

This section describes the specifics of the translation models as delivered for integration. We describe the models which either translate into, or out of English. In particular, the descriptions will cover most of these items:

- Bilingual data used.
- Monolingual data used (for instance, for back-translation (Sennrich et al., 2016b)).
- Language resources used (for instance, part-of-speech taggers, morphological analysers, bilingual dictionaries, machine translation systems etc.)
- NMT platform and architecture
- Training specifics (multi-source input, back-translation, integration of language resources, etc.).
- Indicators of the quality or usefulness of their output.

Automatic evaluation results are reported below for the models as they have been developed by the researchers. We have further, extensive evaluation of these models described in Deliverable 5.4. Initial Progress Report on Evaluation.

A.1 Swahili

A.1.1 Corpora

Tables 1 and 2 describe the parallel and monolingual corpora we used, respectively. As regards parallel corpora, with the exception of GoURMET and SAWA, all of them were downloaded from the OPUS[15] website.[16] We used two additional parallel corpora: the SAWA corpus (De Pauw et al., 2011), that was kindly provided by their editors, and the GoURMET corpus, that was crawled from the web as described by Sánchez-Martínez et al. (2020).

As regards the monolingual corpora, only three monolingual corpora were used: the NewsCrawl (Bojar et al., 2018a) for English, the NewsCrawl for Swahili[17] (Barrault et al., 2019), and the GoURMET monolingual corpus for Swahili. The first two corpora were chosen because they belong to the news domain, the same domain of application of our NMT systems. Given that the size of the Swahili monolingual corpus is much smaller than the size of the English monolingual corpus, additional monolingual data obtained as a by-product of the process of crawling parallel data from the web was used for Swahili.

[15] http://opus.nlpl.eu/
[16] Table 1 contains the parallel corpora available at OPUS at the time of training the systems. New corpora have been added recently, such as the large JW300 corpus (Agić and Vulić, 2019), which we did not use.
[17] http://data.statmt.org/news-crawl/sw/

Corpus	Sentences	en tokens	sw tokens
GoURMET v1	156 061	3 334 886	2 981 699
SAWA	272 544	1 553 004	1 206 757
Tanzil v1	138 253	2 376 908	1 734 247
GV v2017q3	29 698	534 270	546 107
GV v2015	26 033	467 353	476 478
Ubuntu v14.10	986	2 486	2 655
EUbookshop v2	17	191	228
GNOME v1	40	168	170
total	623 632	8 269 266	6 948 341

Table 1: Parallel English–Swahili corpora used to train the NMT systems. *GV* stands for the Global-Voices corpus.

Corpus	Sentences	Tokens
NewsCrawl (en)	40 000 000	796 199 072
NewsCrawl (sw)	414 598	8 377 157
GoURMET (sw)	5 687 000	174 867 482

Table 2: Monolingual Swahili and English corpora used to build synthetic parallel data through back-translation. For the English NewsCrawl corpus, only the size of the subset that has been used for training is displayed.

Preprocesing. Part of the GlobalVoices corpus was reserved for its use as development and test corpora, as described in Section **??**. The remaining sentences from GlobalVoices-v2015 and GlobalVoices-v2017q3, together with the other parallel corpora listed in Table 1 were de-duplicated to obtain the final parallel corpus used to train the NMT systems.

All corpora were tokenised with the Moses tokeniser (Koehn et al., 2007a) and truecased. Parallel sentences with more than 100 tokens in either side were removed. Words were split in sub-word units with byte pair encoding (BPE; Sennrich et al. (2016c)). Table 3 reports the size of the corpora after this pre-processing.

Corpus	Sentences	en tokens	sw tokens
parallel	424 821	7 536 537	6 191 959
NewsCrawl (en)	40 000 000	796 199 072	-
NewsCrawl (sw)	414 598	-	8 377 157
GoURMET mono (sw)	5 687 000	-	174 867 482
development	2 000	41 726	42 037
test (en-sw)	1 863	41 097	41 188
test (sw-en)	1 969	43 149	43 174

Table 3: Size of the corpora used to build the NMT systems after preprocesing.Token counts were calculated before BPE splitting.

A.1.2 Language resources

We *interleaved* (Nadejde et al., 2017) linguistic tags in the target language side of the training corpus with the aim of enhancing the grammatical correctness of the translations.

Morphological taggers were used to obtain the interleaved tags added to the training corpus. The Swahili text was tagged with TreeTagger (Schmid, 2013). We used a publicly available model[18] which was trained on the Helsinki Corpus of Swahili.[19] The English text was tagged with the Stanford tagger (Qi et al., 2018), which was trained on the English Web Treebank (Silveira et al., 2014). While the tags returned by the Swahili tagger were just part-of-speech tags, English tags contained also morphological inflection information. Interleaved tags are removed from the final translations produced by the system.

A.1.3 Model architecture and training

We trained the NMT models with the Marian toolkit (Junczys-Dowmunt et al., 2018). Since hyper-parameters can have a large impact in the quality of the resulting system (Lim et al., 2018; Sennrich and Zhang, 2019), we carried out a grid search in order to find the best hyper-parameters for each translation direction. We explored both the transformer (Vaswani et al., 2017) and recurrent neural network (RNN) with attention (Bahdanau et al., 2014) architectures. Our starting points were the transformer hyper-parameters[20] described by Sennrich et al. (2017) and the RNN hyper-parameters[21] described by Sennrich et al. (2016a).

For each translation direction and architecture, we explored the following hyper-parameters:

- Number of BPE operations: 15 000, 30 000, or 85 000.
- Batch size: 8 000 tokens (trained on one GPU) or 16 000 tokens (trained on two GPUs).
- Whether to tie or not input embedding layers for both languages and output layer.

We trained a system for each combination of hyper-parameters, using only parallel data. Early stopping was based on perplexity on the development set and patience was set to 5 validations, with a validation carried out every 1,000 updates. We selected the checkpoint that obtained the highest BLEU (Papineni et al., 2002) score on the development set.

We obtained the highest BLEU scores on the test set for English–Swahili with an RNN architecture, 30 000 BPE operations, tied embeddings for both languages and single GPU, while the highest ones for Swahili–English were obtained with a Transformer architecture, 30 000 BPE operations, tied embeddings for both languages and two GPUs. This is compatible with the findings by Popel and Bojar (2018), who report that transformer usually performs better with larger batch sizes.

Leveraging monolingual data Once the best hyper-parameters were identified, we tried to improve the systems by making use of the monolingual corpora via back-translation. The quality of a system trained on back-translated data is usually correlated with the quality of the system that

[18]Available at https://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/data/swahili.par.gz
[19]https://korp.csc.fi/download/HCS/a-v2/hcs-a-v2-dl
[20]https://github.com/marian-nmt/marian-examples/tree/master/wmt2017-transformer
[21]https://github.com/marian-nmt/marian-examples/tree/master/training-basics

translates the target language monolingual corpus into the source language (Hoang et al., 2018, Sec. 3). We took advantage of the fact that we are building systems for both directions and applied an iterative back-translation (Hoang et al., 2018) algorithm that simultaneously leverages monolingual Swahili and monolingual English data. It can be outlined as follows:

1. With the best identified hyper-parameters for each direction we built a system using only parallel data.
2. English and Swahili monolingual data were back-translated with the systems built in the previous step.
3. Systems in both directions were trained on the combination of the back-translated data and the parallel data.
4. Steps 2–3 were re-executed 3 more times. Back-translation in step 2 was always carried out with the systems built in the most recent execution of step 3, hence the quality of the system used for back-translation improved with each iteration.

The Swahili monolingual corpus used in step 2 was the GoURMET monolingual corpus. The English monolingual corpus was a subset of the NewsCrawl corpus, the size of which was doubled after each iteration. It started at 5 million sentences and reached 40 million in the fourth iteration, which was the last one.

Since the Swahili NewsCrawl corpus was only made available near the end of the development of our MT systems, it could not be used during the iterative back-translation process. Nevertheless, we added it afterwards: the Swahili NewsCrawl was back-translated with the last available Swahili–English system obtained after completing all the iterations, concatenated to the existing data for the English–Swahili direction and the MT system was re-trained.

A.1.4 Indicators of quality

Table 4 shows the BLEU and chrF++ (Popović, 2017) scores expressed as a percentage, computed on the initial test set, for the different steps in the development of the MT systems. It is worth noting the positive effect of adding monolingual data during the iterative back-translation iterations and that interleaved tags also help to improve the systems according to the automatic evaluation metrics.

A.2 Gujarati

A.2.1 Summary of approach

We used a semi-supervised approach for both translation directions, focusing on the effective use of data other than the small amount of parallel data available, namely (i) English and Gujarati monolingual data and (ii) a Hindi–English parallel corpus. We used data augmentation based on these additional resources to produce synthetic English–Gujarati parallel data, which we used in addition to genuine parallel data to train MT models using the Transformer architecture (Vaswani

Strategy	iteration	BLEU	chrF2++
English–Swahili			
only parallel	-	22.23	46.34
iterative backtranslation	1	25.59	50.08
iterative backtranslation	2	26.22	50.91
iterative backtranslation	3	26.36	51.09
iterative backtranslation	4	26.58	51.39
+ NewsCrawl	4	26.77	51.46
+ NewsCrawl + tags	4	27.42	52.11
Google Translate	-	23.24	48.80
Swahili–English			
only parallel	-	22.66	44.62
iterative backtranslation	1	29.29	51.19
iterative backtranslation	2	29.70	51.82
iterative backtranslation	3	29.99	51.98
iterative backtranslation	4	30.19	52.10
+ tags	4	30.55	52.72

Table 4: Automatic evaluation results for English–Swahili obtained for the different development steps of the MT systems: *only parallel* stands for the systems trained only on parallel data with the best hyper-parameters; *iterative backtranslation* represents systems obtained after iteratively back-translating monolingual data (iteration number is shown in column *iteration*); *+News-Crawl* means that the Swahili NewsCrawl corpus was back-translated and added; and *+ tags* indicates that target language linguistic tags were interleaved.

et al., 2017). More details are provided in the publication describing the models (Bawden et al., 2019) and the models are publicly available online.[22]

Synthetic parallel data was produced from the monolingual data using backtranslation, that is, translating target-side data into the source language (Sennrich et al., 2016b). From the Hindi–English parallel corpus, we produced additional English–Gujarati synthetic parallel data by pivoting through Hindi using a Gujarati–Hindi MT model. To ensure that the three MT models required to produce this data (English→Gujarati, Gujarati→English and Hindi→Gujarati) were of sufficient quality to produce good synthetic data, we used all available data; we trained cross-lingual semi-supervised models for English↔Gujarati and an unsupervised model for Hindi→Gujarati, using cross-lingual language model pre-training of parameters to exploit monolingual data and to boost model performance (Lample and Conneau, 2019).

A.2.2 Data

We use parallel English–Gujarati data, Hindi–English parallel data and monolingual data for each language, available as part of the WMT19 shared task (Barrault et al., 2019). The individual data sources are as follows:

[22] http://data.statmt.org/wmt19_systems/

- English–Gujarati parallel data: Bible, Emille, Govin, WikiTitles v1, Wikipedia, Software data
- Hindi–English parallel data: Emille, Bombay IIT
- English monolingual data: News (news-crawl, news-commentary, news-discuss)
- Gujarati monolingual data: Common crawl, Emille, wiki-dump, News-crawl
- Hindi monolingual data: Bombay IIT, News-crawl

All data sources were used to train the intermediate models used to produce synthetic data. The final models were first trained on both synthetic and genuine parallel data (minus terminologies and software data),[23] and then fine-tuned on the genuine parallel data. Data sizes are given in Table 5.

Training data source	EN→GU	GU→EN
Genuine parallel data	42k	42k
Hindi-to-Gujarati-English parallel data	1.1M	1.1M
Backtranslated monolingual	4.5M	7.1M
Total	5.6M	8.2M

Table 5: Summary of training data used for the English-Gujarati MT models in number of sentences

Data for both languages was normalised and cleaned (training data only, with a maximum sentence length of 80 tokens) using scripts from the Moses toolkit (Koehn et al., 2007b). For English, we tokenised the data using Moses and true-cased it, using a model trained on all news data. Gujarati data was normalised and tokenised using the IndicNLP toolkit (Kunchukuttan, 2020). The data was then segmented into subword units using BPE (Sennrich et al., 2016c) trained with joint vocabularies. BPE vocabularies of size 20,000 were used for all intermediate MT models, and of size 30,000 for all final MT models.

Hindi data was transliterated into the Gujarati script using systematic shifting of Unicode character ranges. This resulted in an increased lexical overlap, in turn aiding cross-lingual learning.

A.2.3 Model architecture and settings

The intermediate MT models were trained using the XLM toolkit (Lample and Conneau, 2019), in which cross-lingual pretraining and semi-supervised MT training is implemented. The models were transformer-based models consisting of 6 encoder layers, 6 decoder layers, 8 heads, with a model/embedding dimension of 512 and feed-forward network dimension of 2048.

The final MT models were trained using the Marian toolkit (Junczys-Dowmunt et al., 2018) and were also transformer-based models. Marian was chosen to train the final models rather than XLM as it offers more training options and is highly optimised, resulting in faster and higher quality models when trained on just parallel data. We used synchronous stochastic gradient descent, a learning rate of 3×10^{-4}, a learning rate warm-up of 16,000 and transformer dropout of 0.1.

[23] We omitted WikiTitles v1 and software data as they contain very short sentences or very out-of-domain sentences.

Our final systems are each ensembles of four models, trained using different random seed initialisations. For English-Gujarati, the best results were achieved by setting the weight of each model in the ensemble manually, guided by the relative BLEU score of each model. We also tried weighting for Gujarati-English, but uniform weights for each in the model in the ensemble produced the best results for this language direction.

A.2.4 Translation quality

The translation quality of the models was initially evaluated in the WMT shared task (Barrault et al., 2019) using both the automatic metric BLEU (Papineni et al., 2002) and human evaluation using direct assessment Graham et al. (2016). Development and test sets in the news domain were produced by professional translators for the shared task. The development set contains data originating in either Gujarati or English, whereas separate test sets were produced for each language direction, whereby the original language of the text corresponds to the source language and the translation to the target language.

As shown in Table 6, the final models achieved 16.4 BLEU (for English→Gujarati) and 21.4 BLEU (for Gujarati→English) on the news-domain test sets produced for the shared task. We also provide results for the intermediate models and after adding each step of our approach (additional data, fine-tuning, ensembling). Out of all constrained systems[24] submitted to the shared task, the GoURMET English→Gujarati model ranked first and the GoURMET Gujarati→English model ranked second out of constrained systems for both the automatic and manual evaluations.

	English→Gujarati		Gujarati→English	
System	Development	Test	Development	Test
Intermediate models	12.6	11.8	22.1	15.5
Final models				
genuine + synthetic data	15.0	14.3	23.8	18.6
+ fine-tuning	16.9	15.1	25.9	20.6
+ Ensemble of 4 models	17.9	16.5	27.2	**21.4**
+ Weighted ensemble	18.1	**16.4**	-	-

Table 6: BLEU scores on the official WMT news dev and test sets (final models marked in bold).

A.3 Turkish

A.3.1 Corpora

All the corpora used for English–Turkish were extracted from Opus (Tiedemann, 2012). Table 7 shows an overview of bilingual resources after pre-processing. Table 8 shows an overview of monolingual corpora after pre-processing. For development and test we used news datasets released by WMT (Bojar et al., 2018b), namely, `newtest2016` and `newstest2017` for development (7,008 sentence pairs) and `newstest2018` for test (3,000 sentence pairs).

[24] Constrained systems are those that use only the data provided for the task and no additional data.

Corpus	Sent's	en tokens	sw tokens
GlobalVoices	5,547	147,203	138,000
SETIMES	207,678	5,988,056	5,763,428
TED2013	137,028	3,043,563	2,918,032
Wikipedia	159,979	6,294,316	6,641,548
Total	510,232	15,473,138	15,461,008

Table 7: Parallel English–Turkish corpora used to train the NMT systems.

Corpus	English		Turkish	
	Sentences	Tokens	Sentences	Tokens
GlobalVoices	1,122,392	29,598,676	6,529	152,755
SETIMES	1,865,812	52,936,420	1,776,431	49,504,944
TED2013	1,904,674	43,330,521	137,013	2,909,617
Wikipedia	10,736,187	360,629,731	175,972	7,371,628
Total	15,629,065	486,495,348	2,095,945	59,938,944

Table 8: Monolingual English and Turkish corpora used to build synthetic data through back-translation.

Pre-processing All corpora were tokenised and truecased using the Moses pipeline (Koehn et al., 2007a). Parallel sentences with more than 100 tokens in either side were removed. Words were segmented in sub-word units with byte pair encoding (BPE; Sennrich et al. (2016c)).

A.3.2 Model architecture and training

We explored attention-based NMT (Bahdanau et al., 2014) as well as auto-encoding variational NMT (AEVNMT; Eikema and Aziz, 2019) developed in WP3 (see D3.1, Task 3.2, Section 3.2.1). AEVNMT is an extension of NMT which models sentence pairs jointly under a latent sentence representation—an embedding of the sentence pair, in this case. AEVNMT has been shown to improve upon standard NMT in mid-resource settings in particular where the training data is a mix of different domains including potentially noisy synthetic data (Eikema and Aziz, 2019). Our NMT models were trained using software developed in WP3.[25]

For each translation direction and architecture, we explored the following hyper-parameters:

- number of BPE merge operations: 8,000 and 14,000;
- conditional training (standard NMT) or joint training (AEVNMT);
- leveraging monolingual data via back-translation or co-training.

[25]https://github.com/Roxot/AEVNMT.pt

To decide on these hyperparameters we used parallel data alone. Models were selected as a function of BLEU (Papineni et al., 2002) score on the development set. The best configuration was attained with $14,000$ independent merge operations for BPE segmentation and with joint modelling. Joint modelling showed a consistent advantage over conditional modelling, which is consistent with the findings of Eikema and Aziz (2019).

Leveraging monolingual data We tried to improve the systems by making use of the monolingual corpora via back-translation as well as a variant of back-translation we refer to as co-training. Co-training is a form of iterative back-translation developed in WP4 (Task 2, see D4.1 Section 3.5), which is better suited for joint generative models. It can be outlined as follows:

1. We train two joint models, one operating source-to-target and another target-to-source.
2. A joint generative model is much like standard NMT, but in addition, it contains a source language model component and it generates both sentences (source and target) from a shared latent space. This latent space is shared across components (language model and translation model) as well as across translation directions (source-to-target and target-to-source).
3. While training we present the model with alternative batches of bilingual, source-monolingual, and target-monolingual data. As expected, both models learn when presented with bilingual data.
4. In a source monolingual batch, the source-to-target model produces a sampled translation into target language in order to provide the target-to-source model with a bilingual sentence pair. In target monolingual batches, something similar happens, but the models flip roles.
5. These alternating roles allow the models to learn even when presented with partial observations (monolingual data).
6. In effect, this looks a lot like iterated back-translation, but where the back-translation step is integrated in training. Thus if a certain sentence is seen multiple times throughout training, each time a potentially different back-translation may be sampled.

A.3.3 Indicators of quality

Table 9 shows BLEU scores (Papineni et al., 2002) using SacreBLEU (Post, 2018) on development and test sets. We report scores for post-processed outputs (i.e., BPE segmentation, tokenization, and truecasing are reversed).[26]

A.4 Bulgarian

A.4.1 Corpora

All the corpora we used for Bulgarian↔English were extracted from Opus (Tiedemann, 2012), ParaCrawl (Bañón et al., 2020) and the WMT monolingual news crawl (Barrault et al., 2019). The test set and the development (held-out) set were both taken from SETIMES2 (in Opus), since it was

[26]Version string: `BLEU+case.mixed+lang.en-tr+numrefs.1+smooth.exp+tok.13a+version.1.4.10`.

	English–Turkish	Turkish–English
Parallel only	11.7	16.1
+ Back-translation	15.4	16.7
Joint + Co-training	15.6	17.1
Test set	14.9	17.7

Table 9: Top: BLEU on development set. Bottom: BLEU on test set using the best setting.

the only news text within our parallel corpora. The total size of the parallel data on Opus is about 44 million sentence pairs, with about 40 million of these coming from the OpenSubtitles corpus, and to this we added about 1,000,000 sentence pairs from ParaCrawl (v4).[27] For the monolingual Bulgarian news we took all available data (about 35 million sentences) whereas for English we randomly sampled 50 million sentences from the 2016–2018 news crawls. We show the exact sizes of the corpora (after filtering out any training sentences which matched those in the development or test sets) in Table 10.

Corpus	Sentences	en tokens	bg tokens
Opus train (w/o OpenSubtitles)	3 817 861	75 007 031	74 383 117
OpenSubtitles	40 203 254	342 731 883	286 886 158
ParaCrawl train	1 039 885	24 419 846	23 398 209
News crawl (en)	49 991 216	1 160 074 749	–
News crawl (bg)	35 006 166	–	704 370 934
SETIMES2 dev	2 000	49 562	50 383
SETIMES2 times	2 000	49 207	49 091

Table 10: Size of the corpora used to build the Bulgarian↔English systems. The OpenSubtitles data is contained in Opus, but we show it broken out since we performed experiments with and without this part of the data. Token counts are for tokenised text, before BPE splitting.

The preprocessing of the data was a standard pipeline of normalisation, tokenisation and truecasing using Moses (Koehn et al., 2007a) followed by BPE (Sennrich et al., 2016c) with models trained separately on each side of the Opus parallel data, and 50,000 merges.

A.4.2 Model architecture and training

All experiments were performed with Marian (Junczys-Dowmunt et al., 2018). First we built a shallow RNN-based model for back-translation, and translated all the monolingual sentences with length less than 150, using a beam size of 5.

For the main experiments, we tried both both deep (4-layer) RNN models and transformer (regular) models (Vaswani et al., 2017), choosing the latter for our deployed system. We applied layer normalisation and label smoothing (0.1) with word dropout and transformer dropout also set to 0.1.

[27] https://www.paracrawl.eu/v4

For the final system we did not use OpenSubtitles as this reduced BLEU slightly, but used all the Opus parallel data plus ParaCrawl, as well as the synthetic back-translated data. When adding the synthetic data, the parallel data was over-sampled 10× for bg–en and 8× for en–bg to give roughly a 1:1 ratio between synthetic and parallel data.

A.4.3 Indicators of quality

We show BLEU scores for different data conditions in Table 11. The first thing to notice is that the scores are very high, and there is little variation in the scores. This likely reflects the fact that SETIMES2 is quite uniform in topic and style, so even though we removed any exact matches between test and train, there is still a close similarity. Beyond that, we note that the transformer and deep RNN have similar performance, and we found in other experiments (not shown) that transformer dropout was vitally important when over-sampled data was included in training. The addition of synthetic news data did not have a clear positive effect on performance, but we included it in the final system since our target is the news domain. We suspected that the lack of discernment in the test setup could be a reason for the lack of clear gain from the synthetic news data.

	Deep RNN		Transformer Regular	
	bg-en	en-bg	bg-en	en-bg
Only parallel	50.5	48.1	49.8	48.9
+ Open Subtitles	48.5	46.5	48.5	46.5
+ Backtranslation, drop-out	51.5	49.5	51.7	47.7
+ Paracrawl, normalise	–	–	51.7	48.5

Table 11: Comparison on Bulgarian↔English. We show BLEU scores on our test portion of the SETIMES2 corpus

A.5 Tamil

A.5.1 Summary of approach

Similar to the approach used for the English–Gujarati and Gujarati–English models (see Section A.2, we explore the use of pre-training and data augmentation for the English–Tamil and Tamil–English models.

We train a range of different models on the data available (Table 12) and exploit backtranslations from these models to iteratively train subsequent models, also exploring how best to combine the different sources of backtranslated data.

A.5.2 Data

We use available parallel and monolingual data made available for the WMT2020 shared task on news translation. This includes parallel and monolingual English–Tamil data, as well as data for English–German, which we use to pre-train one of our models.

All data was cleaned to filter poor examples, keeping sentences with a minimum length of 3, a maximum length of 100, those for which the length ratio between the parallel sentences is maximum 2.2, those that do not contain 50% non-alphabetic characters or more than 50% of words without an alphabetic character.footnoteAn alphabetic character being defined as one belonging to the language in question: the Latin alphabet for English and the Tamil script, which is an abugida script. We deduplicate the data and normalise using Moses (Koehn et al., 2007b). We then apply subword segmentation using SentencePiece (Kudo and Richardson, 2018), jointly learnt on both English and Tamil.

Data type	#sentences	Corpora
Parallel en–ta	340,995	PMindia, Tanzil, NLCP, PIB, MKB, EnTam
Monolingual en (in-domain)	653,606,835	News (crawl, discussions, commentary)
Monolingual en (out-of-domain)	101,692,093	Europarl, Wiki dumps
Monolingual ta (in-domain)	668,008	News crawl
Monolingual ta (out-of-domain)	1,553,160	Wiki dumps
Parallel de–en	43,675,462	Europarl, News commentary, Paracrawl, WikiMatrix, Tilde Rapid

Table 12: Data used for the Tamil-English models. Note that we also use German-English data for some of our experiments as a form of pretraining.

A.5.3 Models tested

Having trained different types of models with different pretraining objectives, we opt to concentrate on two different models to perform iterative backtranslation:

1. **Pretrained mBART models**: transformer-base models trained using the Marian toolkit (Junczys-Dowmunt et al., 2018) pretrained on the mBART language modelling objective (Liu et al., 2020), corresponding to learning to correct perturbations in the data in a sequence to sequence fashion.
2. **Pretrained de–en models**: a Marian transformer-base model pretrained on translation for an unrelated but higher resource language pair. We choose to pretrain on de–en translation, using a vocabulary that is shared between English, German and Tamil, in order for those parameters to be fine-tuned afterwards on English-Tamil translation.

Our final models are the product of iterative backtranslation using these two types of system, in combination with multi-agent dual learning (MADL). The process is summarised visually in Figure 9, where each block represents an MT model and the arrows represent the production of backtranslated data to be used as training data for the next model, and we provide some details about the backtranslation and MADL steps below.

Backtranslation The backtranslation step represents the production of synthetic parallel data by translating monolingual in-domain data using an intermediate MT model. The resulting backtranslations were cleaned to remove poor quality translation pairs and those for which the length ratio is

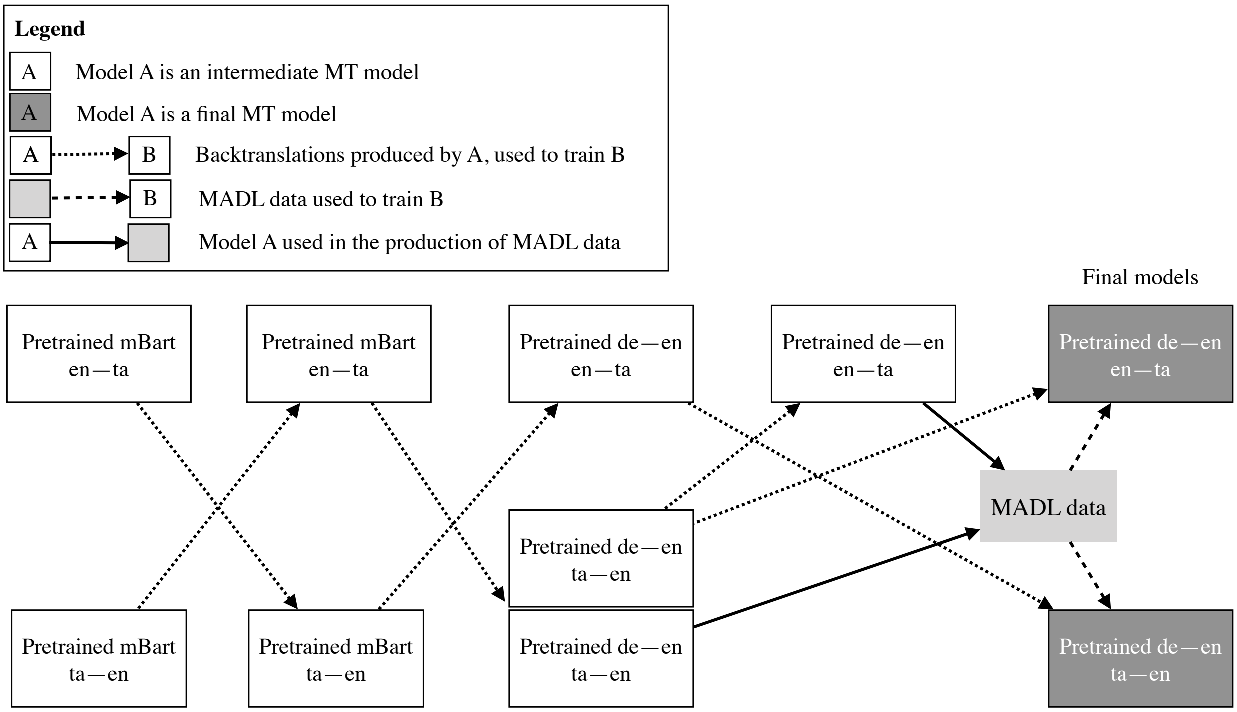

Figure 9: Schema of the iterative backtranslation process we used to produce our final en–ta models. Final single models are shown in grey.

too large and then scored using dual conditional cross-entropy filtering (Junczys-Dowmunt, 2018) so that they could be ranked according to their quality. We took the top ranked backtranslations for each direction (5M originally en, all originally ta)[28] to be used to train the model in the next step. When training on these backtranslations, we used a combination of authentic and backtranslated data, and then fine-tune on the authentic parallel data.

MADL data We created additional synthetic data from the parallel data using an adaptation of Multi-Agent Dual Learning (Wang et al., 2019; Kim et al., 2019). Using ta–en and en–ta models, we created forward- and backtranslations of the parallel data using 6-best lists to select the best translations based on BLEU. We also created backtranslations of the target sentences for each direction. MADL data in Figure 9 is a combination of this synthetic data and the true parallel data.

A.5.4 Translation quality

We report the BLEU scores of our models in Table 13. We show results on the WMT dev set as well as a test set produced from BBC data. We do not currently have access to the WMT test set, as the reference side of the test set has not yet been released publicly. We provide a baseline result of a Marian model trained on the parallel data alone in the first row, showing the gain that can be achieved through pretraining and iterative backtranslation.

[28]We use all Tamil sentences given the small number of them.

	en–ta		ta–en	
Model	WMT dev	BBC test	WMT dev	BBC test
Marian base	5.1	-	10.1	-
Pretrained de-en final models	12.59	11.63	22.21	28.60

Table 13: A summary of the BLEU score results of the models.

A.6 Serbian

This section describes the process of building the Serbian-English and English-Serbian models delivered for integration.

Corpus	Sentences	en tokens	sr tokens
Opus	17340943	157618264	139532584
WikiMatrix	354928	7242512	6597796
GoURMET SR	304296	9932281	8646256
GoURMET BSH	334095	9407817	8288598
GoURMET DW	76705	1747265	1603460
News domain total	300209	6570243	6179076
All parallel total	18410967	185948139	164668694

Table 14: Parallel corpora used for the En-Sr systems. Corpus statistics are shown after cleaning the corpora.

A.6.1 Corpora

The corpora used for the En-Sr systems are shown in Tables 14 (parallel) and 15 (monolingual). The corpus statistics shown are of the corpora after cleaning (described below). The parallel data we used was from Opus (Tiedemann, 2012), WikiMatrix (Schwenk et al., 2019), and GoURMET crawled corpora. We also used monolingual data from the news domain from NewsCrawl[29] (Barrault et al., 2019) and GoURMET crawled data from the Deutsche Welle website.

The GoURMET crawled corpora were collected using Bitextor (Espla-Gomis and Forcada, 2010) to crawl websites for parallel data and to extract parallel and monolingual data from Deutsche

[29] http://data.statmt.org/news-crawl/en/ and http://data.statmt.org/news-crawl/sr/

Corpus	Sentences	Tokens
NewsCrawl (sr)	3510550	70127018
DW crawl (sr)	1025945	21492627
NewsCrawl (en)	190769622	3921134020
DW crawl (en)	4048203	88159024

Table 15: Monolingual corpora used for the En-Sr systems. The corpus statistics are shown after cleaning the corpora.

Welle dumps. The creation process is described in more detail in D1.2.

A.6.2 Preprocessing

All parallel data was cleaned to only contain sentences with a minimum number of 3 and maximum number of 100 words and a source to target sentence length ratio over 0.6 and under 1.66. Sentences containing too many characters that are not in the respective language's script were excluded. The data was further cleaned using language identification with CLD2 [30], with the Serbian side permitting sentences recognised as Bosnian or Croatian due to the similarity between the three languages. Sentences were also deduplicated and cleaned to remove leading dashes, bullet points and closing brackets, also opening brackets at the end of the sentence. All Cyrillic Serbian data was transliterated into the Latin script. We applied BPE segmentation using SentencePiece (Kudo and Richardson, 2018). We trained a joint SentencePiece model on all parallel and equal amounts of monolingual data for both Serbian and English.

A.6.3 Model architecture and training

We built Transformer models (Vaswani et al., 2017) using the Marian NMT toolkit (Junczys-Dowmunt et al., 2018). We initially trained baseline models using only parallel data. These models were then used to backtranslate in-domain monolingual data (Sennrich et al., 2016b).

We then finetuned the initial models using both parallel and backtranslated monolingual in-domain data. The parallel data used for finetuning consisted of the SETIME2,GlobalVoices and DW crawl corpora, with the DW crawl corpus oversampled three times. For the Serbian→English model, the backtranslated data we used consisted of all DW monolingual crawled data as well as 2.5 million sentences from News crawl, a total of 6.5 million sentences. For the English→Serbian model, we used all of the Serbian News crawl and oversampled the DW crawl monolingual data 3 times, resulting in about 6.5 million sentences. For both models the parallel data was oversampled 15 times so that the ratio of true parallel to synthetic data is about 1:1.

A.6.4 Indicators of quality

Table 16 shows BLEU scores on the DW test set. Finetuning on the in-domain data improves performance over the baseline although not by a large margin.

	sr-en	en-sr
baseline	27.6	20.5
finetuned on in-domain	28.6	21.8

Table 16: Automatic evaluation results for English-Serbian on an automatically aligned test set from DW data.

[30] https://pypi.org/project/pycld2/

A.7 Amharic

A.7.1 Corpora

We used the GoURMET parallel corpus (as described in D1.2) which consists of 58,124 sentence pairs. In order to create development and test sets, we randomly sample 3,000 sentences for each set. The remaining sentences are used for the training set. Furthermore, we used the GoURMET monolingual corpus for Amharic which contains about 3 million sentences. For English, we randomly sub-sampled a dataset of similar size from the NewsCrawl 2019 monolingual corpus.[31] (Barrault et al., 2019) A description of the different corpora can be found in Table 17.

The GoURMET parallel corpus involved crawling and aligning parallel websites which was done by using by Bitextor (Espla-Gomis and Forcada, 2010). The resulting raw parallel corpus was then further processed with Bicleaner (Sánchez-Cartagena et al., 2018). More details can be found in deliverable D1.2.

	Sentences	English tokens	Amharic tokens
Training	52,124	1,019,180	801,282
Development	3,000	57,484	45,431
Test	3,000	58,500	46,199
GoURMET monolingual (am)	3,251,888	-	46,745,049
NewsCrawl monolingual (en)	3,303,968	89,710,775	-

Table 17: Number of sentences and tokens for the different available data after pre-processing but before applying BPE.

Pre-processing The data is lowercased and tokenized. In addition, Amharic is tokenized by splitting on punctuation, which includes specific word-boundary markers.[32] We used Moses (Koehn et al., 2007b) to tokenize and to learn a recaser over the English side of the training set. Moreover, sentences in the monolingual corpora that contained more than 100 tokens were removed. Words were segmented in sub-word units with byte-pair encoding (BPE; Sennrich et al. (2016c)) where we used separate sub-word vocabularies.

A.7.2 Model architecture and training

For the experiments we explored NMT and SMT systems. The NMT systems were trained using Fairseq (Ott et al., 2019). After a fairly extensive hyperparameter search we decided for Transformer models (Vaswani et al., 2017) over recurrent ones (Bahdanau et al., 2014). Some hyperparameters seemed crucial and that included layer normalization, dropout (0.2), and label-smoothing (0.2). We early stop training after 5 checkpoints without improvement on development BLEU (Papineni et al., 2002).

The SMT systems were trained using Moses (Koehn et al., 2007b). The default parameters were used except that word alignment was done with fast_align (Dyer et al., 2013) as opposed to

[31] http://data.statmt.org/news-crawl/en/

[32] The list of Amharic punctuation signs is taken from https://en.wikipedia.org/wiki/Amharic#Punctuation

GIZA++ (Och and Ney, 2003). In addition, we experimented with adding monolingual data to the target side in order to learn a 5-gram language model (Heafield, 2011; Heafield et al., 2013) which is used as an additional feature. We explored two SMT systems: phrase-based SMT (PBSMT) and a hierarchical phrase-based system namely Hiero (Chiang, 2005). For experiments involving monolingual data, we used the available data after pre-processing.

Approach taken The final model is a NMT system that was trained on a combination of parallel data and back-translated data that was produced by a SMT system. We can describe the final model in the following steps:

1. Train a PBSMT system on the parallel data which in addition incorporates monolingual data on the target side.
2. Use the trained PBSMT system to backtranslate Sennrich et al. (2016b), that is, to obtain synthetic data.
3. Train the NMT system where the parallel data is up-sampled to be 10% of the data and the rest to be synthetic data.
4. After convergence, fine-tune solely on the parallel data.

Furthermore, we also conducted experiments where the synthetic data was produced by a NMT system however this gave a poor performance. In addition, we noticed that, when training a NMT system with synthetic data, it did not make a difference whether the synthetic data was produced by a phrase-based or a hierarchical SMT system.

A.7.3 Test Sets

The development and test set for English-Amharic were obtained from the GoURMET English-Amharic crawled parallel corpus. There was no provided split for the evaluation sets and therefore we randomly sample sentences. We sample randomly 3,000 unique sentences for each evaluation set.

A.7.4 Indicators of quality

Table 18 shows BLEU scores on the test set. We can see that SMT with monolingual data at the target side outperforms NMT trained on parallel data. However, NMT trained with a combination of parallel data and synthetic data performs the best overall.

A.8 Kyrgyz

This section describes the resources used and the steps followed to build the English–Kyrgyz NMT systems for both translation directions. Given the limited availability of parallel data for this language pair, a combination of multilingual machine translation (Johnson et al., 2017), pivoting (Huck and Ney, 2012) and backtranslation was applied with the aim of leveraging other sources of information.

	English-Amharic	Amharic-English
PBSMT	9.0	10.9
Hiero	10.6	11.3
NMT	8.3	10.6
+ synthetic PBSMT	**12.1**	**15.9**

Table 18: BLEU scores on the test set for the different models. Final models are displayed in bold.

A.8.1 Corpora

Tables 19 and 20 show the parallel English–Kyrgyz and monolingual corpora used, respectively.

As regards parallel corpora, all the corpora available from the OPUS[33] website was used together with one additional parallel corpus: the GoURMET corpus, which was crawled from the web following the method described in deliverable D1.2. That method involved identifying and crawling parallel websites from the top-level domain `.kg`, processing them with Bitextor (Espla-Gomis and Forcada, 2010) and filtering the resulting parallel sentences with Bicleaner (Sánchez-Cartagena et al., 2018).

Concerning monolingual corpora, only three corpora were used: the NewsCrawl (Bojar et al., 2018a) for English, the NewsCrawl for Kyrgyz,[34] and the GoURMET monolingual corpus for Kyrgyz. The first two corpora were chosen because they belong to the news domain, the same domain of application of the NMT systems built. Given that the size of the Kyrgyz monolingual corpus is much smaller than the size of the English monolingual corpus, additional monolingual data obtained as a by-product of the process of crawling parallel data from the web was used for Kyrgyz.

Parallel corpora for other language pairs were also used by means of multilingual machine translation and pivoting. These pairs were English–Kazakh, English–Russian and Kyrgyz–Russian, and the corpora are listed in Table 21. For English–Kazakh, the parallel corpora built by Sánchez-Cartagena et al. (2019) for their submission to the WMT 2019 News Translation shared task was reused. Since they carried out iterative backtranslation, there are two different corpora: one for each direction, and both of them contain synthetic data. For English–Russian, a subset of the United Nations parallel corpus (Ziemski et al., 2016) was chosen, and for Kyrgyz–Russian, a parallel corpus was also crawled from the web following the same strategy applied for obtaining English–Kyrgyz data.

Preprocesing. Part of the GoURMET crawled parallel corpus corpus was reserved for its use as development and test corpora, as described in Section **??**. The JW300 corpus was processed to remove sentences written in a different language or containing old English. The scripts used have been released and are available at https://github.com/transducens/gourmet-ua/tree/master/jw. The sentences from the GoURMET crawled parallel corpus which were not used for the test or the development set and the result of cleaning the JW300 were concatenated with the other parallel corpora listed in Table 19 and de-duplicated in order to obtain the English–Kyrgyz parallel corpus used to train the NMT systems.

[33] http://opus.nlpl.eu/
[34] http://data.statmt.org/news-crawl/ky/

Corpus	sentences	en tokens	ky tokens
GoURMET	63 306	840 680	640 600
JW300 v1	342 868	5 915 419	4 835 011
Ubuntu v14.10	10 087	25 656	24 210
GNOME v1	25 458	92 944	81 255
wikimedia v20190628	36	7 457	7 140
QED v2.0a	1 501	23 324	17 729
Tatoeba v20190709	114	566	416

Table 19: Parallel English–Kyrgyz corpora used to train the NMT systems.

Corpus	sentences	Tokens
NewsCrawl (en)	18 113 311	359 823 264
NewsCrawl (ky)	279 440	3 420 809
GoURMET (ky)	846 048	11 281 326

Table 20: Monolingual Kyrgyz and English corpora used to build synthetic parallel data through back-translation.

Corpus	pair	sentences	SL tokens	TL tokens
GoURMET	Russian–Kyrgyz	261 921	3 162 747	3 174 648
WMT 2019	English→Kazakh	11 856 332	225 713 404	180 830 862
WMT 2019	Kazakh→English	20 027 419	312 134 093	398 286 113
UN	English–Russian	23 239 280	524 719 605	482 966 738

Table 21: Parallel corpora from other language pairs used to train the NMT systems.

Corpus	Languages	sentences	SL tokens	TL tokens
parallel	English–Kyrgyz	336 630	5 244 646	3 874 527
NewsCrawl	English	20 000 000	397 471 088	
NewsCrawl	Kyrgyz	279 440	3 420 809	
GoURMET mono	Kyrgyz	846 048	11 281 326	
GoURMET	Russian–Kyrgyz	95 314	1 220 094	1 244 545
WMT 2019	English→Kazakh	11 856 332	225 713 404	180 830 862
WMT 2019	Kazakh→English	20 027 419	312 134 093	398 286 113
UN	English–Russian	10 874 279	300 491 012	275 448 555
development	English–Kyrgyz	2 255	34 740	27 367
test	English–Kyrgyz	1 261	22 310	17 579

Table 22: Size of the corpora used to build the NMT systems after preprocesing. English-to-Kazakh and Kazakh-to-English corpora have different sizes because they were obtained after back-translating monolingual data. For the English NewsCrawl corpus, only the size of the subset that has been used for training is displayed. Token counts were calculated before BPE splitting.

Concerning parallel corpora for other language pairs listed in Table 21, the UN parallel corpus was filtered so as to ensure that it only contained sentences starting with an uppercased character and ending with a dot, and the GoURMET crawled Russian–Kazakh parallel corpus was filtered with Bicleaner using a threshold of 0.65.

All corpora were tokenized with the Moses tokenizer (Koehn et al., 2007a) and truecased. Parallel sentences with more than 100 tokens in either side were removed. Words were split in sub-word units with byte pair encoding (BPE; Sennrich et al. (2016c)). Table 22 reports the size of the corpora after this pre-processing.

A.8.2 Model architecture and training

The NMT models were trained with the fairseq toolkit (Ott et al., 2019). Since training hyper-parameters can have a large impact in the quality of the resulting system (Lim et al., 2018; Sennrich and Zhang, 2019), a grid search was carried out in order to find the best hyper-parameters for each translation direction. Both the Transformer (Vaswani et al., 2017) and recurrent neural network (RNN) with attention (Bahdanau et al., 2014) architectures were explored. The starting points were the Transformer hyper-parameters[35] described by Sennrich et al. (2017) and the RNN hyper-parameters[36] described by Sennrich et al. (2016a).

Firstly, the best set of hyperparameters for the baseline systems which were trained solely on the available parallel data for English–Kyrgyz were determined. The following hyperparameters were explored for each translation direction and architecture:

- Number of BPE operations: 5 000, 10 000, 20 000, 40 000 or 80 000.

[35] https://github.com/marian-nmt/marian-examples/tree/master/wmt2017-transformer
[36] https://github.com/marian-nmt/marian-examples/tree/master/training-basics

- Batch size: 3 000 tokens (trained on one GPU), 6 000 tokens (trained on two GPUs) or 9 000 tokens (trained on three GPUs).
- Whether to tie or not decoder input and output embeddings (Press and Wolf, 2017).
- Model size. For RNN systems, we explored the following hidden/embedding size pairs: 1024/512, 512/512 and 256/256. For Transformer systems, we explored the following model sizes: 512, 256, 128.

Afterwards, grid search was repeated for multilingual systems, whose training data is described later in this section. Given the training overhead caused by the addition of large amounts of data from other language pairs, only the transformer and RNN starting points were compared, using 1 GPU and 30 000 BPE operations in both cases.

In all cases, early stopping was based on perplexity on the development set and patience was set to 10 validations, with a validation carried out every $1,000$ updates for systems trained only English–Kyrgyz parallel data and every 5 000 updates for multilingual systems.

For the systems trained only on parallel corpora, the best performing systems for both directions followed the RNN architecture with the largest model size and 10 000 BPE operations. Tied embeddings were effective only for the Kyrgyz–English direction. For multilingual systems, the Transformer architecture was the most effective one.

Leveraging additional data Baseline systems trained solely on the available parallel data were improved by making use of the additional data described previously: monolingual corpora were leveraged via back-translation and parallel corpora from other language pairs via multilingual MT and pivoting. Since the quality of a system trained on backtranslated data is usually correlated with the quality of the system that translates the TL monolingual corpus into the SL (Hoang et al., 2018, Sec. 3), the systems were trained by following these steps, which are summarized in Figure 10.

1. Train systems using a combination of English–Kyrgyz parallel corpora and parallel corpora from other language pairs
2. Backtranslate (Sennrich et al., 2016b) the monolingual data
3. Train the final systems on all the resources available

For the first step, the GoURMET Russian–Kyrgyz parallel corpus was converted into an English–Kyrgyz corpus by translating the Russian sentences into English with a Transformer system trained on all the English–Rusian data available for the WMT 2019 news translation shared task. This corpus was concatenated to the parallel corpus described in Table 22. Then, the multilingual systems were trained. The corpora and their proportion in the training set were different for each direction. For English-to-Kyrgyz, the multilingual system was trained on the concatenation of English–Kyrgyz and English–Kazakh data. For the opposite direction, three language pairs were combined: Kyrgyz–English, Kazakh–English and Russian–English. In both cases, English–Kyrgyz data was oversampled to approximately match the size of the other language pairs. The Kazakh–English corpus extracted from the WMT 2019 submission by Sánchez-Cartagena et al. (2019) was *undersampled* instead: only a half of it, chosen randomly, was used. Table 23 summarizes the multilingual training data.

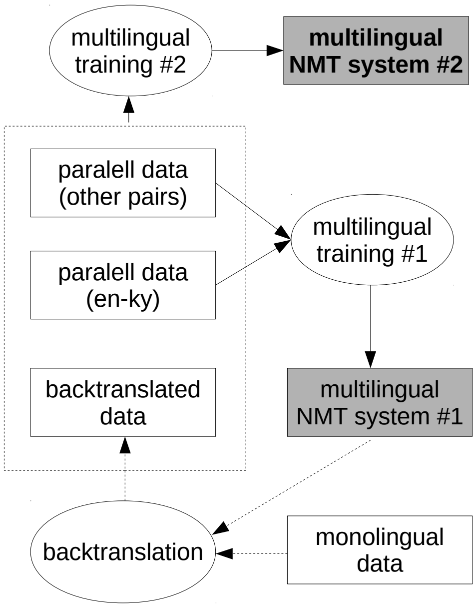

Figure 10: Steps followed to train the final English-to-Kyrgyz and Kyrgyz-to-English systems. The final system is highlighted in bold.

Language pair	Size (sentences)	oversampling ratio
English→Kyrgyz		
English–Kyrgyz	408 220	30×
English–Kazakh	11 856 332	1×
Kyrgyz→English		
English–Kyrgyz	408 220	30×
Kazakh–English	20 027 419	0.5×
Russian–English	10 874 279	1×

Table 23: Data used for training multilingual systems.

Language pair	Size (sentences)	oversampling ratio
English→Kyrgyz		
English–Kyrgyz	1 519 555	6x
English–Kazakh	11 856 332	1x
Kyrgyz→English		
English–Kyrgyz	20 349 082	1x
Kazakh–English	20 027 419	0.5x
Russian–English	10 874 279	1x

Table 24: Data used for training multilingual systems, after backtranslating monolingual corpora. The rows labelled as English–Kyrgyz account for the concatenation of the genuine parallel and backtranslated data.

Strategy	BLEU	chrF++
English→Kyrgyz		
only parallel	4.26	22.99
+ pivot Russian–Kyrgyz	5.16	25.64
+ multilingual	8.53	33.50
+ backtranslation	11.45	38.47
Google Translate	7.02	32.24
Kyrgyz→English		
only parallel	7.70	28.07
+ pivot Russian–Kyrgyz	9.07	30.24
+ multilingual	16.31	41.02
+ backtranslation	18.18	42.80
Google Translate	15.13	41.29

Table 25: Automatic evaluation results for English–Kyrgyz obtained for the different development steps of the MT systems.

The systems from the first step were used to backtranslate the monolingual data described in Table 22. Then, the final multilingual systems were trained on the same language pair combinations as those in the first step. Given that the addition of backtranslated data made the English–Kyrgyz data larger, oversampling ratios changed. This information is depicted in Table 24, where the English–Kyrgyz figures represent the result of concatenating the original parallel data and the backtranslated data.

A.8.3 Indicators of quality

Table 25 shows the BLEU (Papineni et al., 2002) and chrF++ (Popović, 2017) scores expressed as a percentage, computed on the test set, for the different steps in the development of the MT systems. It is worth noting the positive effect of all forms of additional data leveraged: parallel data from other language pairs via pivoting and multlingual MT, and monolingual data via backtranslation.

ENDPAGE

GoURMET

H2020-ICT-2018-2 825299

D5.3 Initial Integration Report